Interleaving Natural Language Parsing and Generation Through Uniform Processing

Günter Neumann

**Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH**

**Research
Report**
96-03

Interleaving Natural Language Parsing and Generation Through Uniform Processing

Günter Neumann

March 1996

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Sema Group, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry of Education, Science, Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct systems with technical knowledge and common sense which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Computer Linguistics
- Programming Systems
- Deduction and Multiagent Systems
- Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Dr. Dr. D. Ruland

Director

Interleaving Natural Language Parsing and Generation Through Uniform Processing

Günter Neumann

DFKI-96-03

This work has been supported by a grant from The Federal Ministry of Education, Science, Research and Technology (FKZ ITWM-9403).

ISSN 0946-008X

Interleaving Natural Language Parsing and Generation Through Uniform Processing

Günter Neumann

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Stuhlsatzenhausweg 3
66123 Saarbrücken, Germany
neumann@dfki.uni-sb.de

Abstract

We present a new model of natural language processing in which natural language parsing and generation are strongly interleaved tasks. Interleaving of parsing and generation is important if we assume that natural language understanding and production are not only performed in isolation but also can work together to obtain subsentential interactions in text revision or dialog systems.

The core of the model is a new uniform agenda-driven tabular algorithm, called $\mathcal{UTA}$. Although uniformly defined, $\mathcal{UTA}$ is able to configure itself dynamically for either parsing or generation, because it is fully driven by the structure of the actual input—a string for parsing and a semantic expression for generation.

Efficient interleaving of parsing and generation is obtained through *item sharing* between parsing and generation. This novel processing strategy facilitates exchanging items (i.e., partial results) computed in one direction automatically to the other direction as well.

The advantage of $\mathcal{UTA}$ in combination with the item sharing method is that we are able to extend the use of memoization techniques even to the case of an interleaved approach. In order to demonstrate $\mathcal{UTA}$'s utility for developing high-level performance methods, we present a new algorithm for *incremental self-monitoring* during natural language production.

1 Introduction

In the area of natural language processing in recent years, there has been a strong tendency towards *reversible natural language grammars*, i.e., the use of one and the same grammar for grammatical analysis (parsing) and grammatical synthesis (generation) in a natural language system.

The idea of representing grammatical knowledge only once and of using it for performing both tasks seems to be quite plausible, and there are many arguments based on practical and psychological considerations for adopting such a view (e.g., [Frazier, 1982; Kempen and Hoenkamp, 1987; Jacobs, 1988; Shieber, 1988; Appelt, 1987; Alshawi and Crouch, 1992; VanNoord, 1993; Ristad, 1993]). Recent developments in the area of constraint-based grammar theories—due to their declarative and formal status—demonstrate that grammar reversibility is in fact computationally feasible.

Nevertheless, in almost all large natural language systems in which parsing and generation are considered in similar depth, different algorithms are used—even when the same grammar is used.

At present, the first attempts are being made at *uniform* architectures which are based on the paradigm of natural language processing as deduction [Pereira and Warren, 1983], [Shieber, 1988]. Here, grammatical processing is performed by means of the same underlying deduction mechanism, which can be parameterised for the specific tasks at hand.

Natural language processing based on a uniform deduction process has a formal elegance and results in more compact systems. There is one further important advantage that is of both theoretical and practical relevance: a uniform architecture offers the possibility of viewing parsing and generation as *strongly interleaved tasks*. Interleaving of parsing and generation is important if we assume that natural language understanding and production are not performed in isolation but rather can work together to obtain a flexible use of language.[1] In particular this means

1. the use of one mode of operation for monitoring and controlling the other, and

2. the use of structures resulting from one direction directly in the other.

For example, during generation integrated parsing can be used to monitor the generation process and to cause some kind of revision, e.g., *to reduce the risk of misunderstandings*. Research on monitoring and revision strategies is a very prominent area in cognitive science (cf. [Berg, 1986; Levelt, 1989]); however, currently there exists no algorithmic model of such a behaviour. A uniform architecture can be an important step in that direction. Further attractive applications for interleaved parsing and generation are the exploration of highly interactive text-processing facilities such as structure-editing operations, propagation of minimal grammatical changes [Wirén and

[1] We use the terms parsing and generation only for grammatical processing. Thus if we refer to the whole task of language processing we use the terms understanding and production.

Rönnquist, 1993], grammar and style checkers, on-line translation, in which the target-language text is generated in parallel with the source-language text [Somers *et al.*, 1990], and text revision [Vaughan and McDonald, 1986]. Interleaved parsing and generation seems also promising for question-answering systems where question understanding and answering is performed simultaneously [Robertson, 1994] and for bidirectional dialogue systems [Levine, 1992].

Modelling such high-level performance methods on the basis of non-uniform approaches is problematic—if not impossible. For example, if two different grammars and algorithms are in use then additional translation operations are necessary in order for parsing and generation to exchange partial results. Since, this is a complex process in itself, not even maintaining two specific grammars but also two different algorithms will be a handicap for an interleaved approach.

Unfortunately, the currently proposed uniform architectures are too inflexible and inefficient so that it seems unclear how an efficient task-oriented uniform model could be achieved. An obvious problem is that different input structures are involved in each direction—a string for parsing and a semantic expression for generation—which causes a different traversal of the search space defined by the grammar. Even if this problem were solved, it is not that obvious how a uniform model could re-use partial results computed in one direction efficiently in the other direction for obtaining a practical interleaved approach to parsing and generation.

The contribution of this work In this paper we present a novel uniform algorithm (called $\mathcal{UTA}$) for parsing and generation of constraint-based grammars that overcomes these problems. The most interesting properties of $\mathcal{UTA}$ are:

1. a uniform data-driven processing strategy,
2. item sharing between parsing and generation, and
3. co-routine relationship between parsing and generation.

The first property means that parsing and generation are both realized by the single program $\mathcal{UTA}$ but that it is able to configure itself dynamically for either parsing or generation. The only *essential parameter* for $\mathcal{UTA}$ to adapt itself efficiently to either the parsing or generation task is the feature that carries the input, hence we call it the *essential feature* (Ef). This information suffices to define a data-driven selection function (Ef determines the selection of the next right hand side element of a rule), and a uniform chart mechanism (partial results are ordered according to the value of their Ef).

Secondly, $\mathcal{UTA}$ extends the traditional usage of a chart by allowing for *shared items* between parsing and generation: Partial results computed in one direction are automatically made available for the other direction as well. Thus, if parsing and generation work in tandem to solve some specific problem they are capable to exchange the result

of partial computations, which reduces the amount of unnecessary computations in those cases. In other words, $\mathcal{UTA}$ extends the usage of a chart even for the case that parsing and generation are strongly interleaved.

Interleaving of parsing and generation is realized using a *co-routine processing* regime between both directions using a flexible agenda mechanism. Here, parsing and generation are considered as specific instances of $\mathcal{UTA}$. We call the different instances *parser* and *generator* (but note, both are realized by the same algorithm). The only differences are 1.) different values for the essential feature Ef, and 2.) each one has its own private agenda. The agenda control—which is the same for both—is able to co-routine between both directions in a *fine-grained incremental* manner. For example, during parsing generation is called for a just analysed partial string. The result of the generator may then influence parsing of next partial strings. Obviously, this complex processing strategy benefits directly from the item sharing mechanism introduced above. As another example, we show in detail how integrated parsing is used during generation for *incremental self-monitoring* of the generation process. It will turn out that such a complex process can be realized quite easily and efficiently using $\mathcal{UTA}$'s novel properties.

$\mathcal{UTA}$ as well as the incremental monitoring strategy have been fully implemented in Common Lisp and CLOS and tested with constraint-based lexicalized grammars for Dutch and German. It uses the powerful constraint-solver $\mathcal{UDiNe}$ which is capable of dealing with distributed disjunctions over arbitrary structures, negative co-references, and full negation [Backofen and Weyers, 1993].

Overview of the next sections In the next section we introduce the formal and linguistic background on which our approach is based. Especially we introduce abstractly the notions of reversible grammar and uniform algorithm, and introduce *constraint logic programming* (CLP) as an appropriate means for establishing the computational basis for uniform processing. In section 3 we describe in detail the new uniform tabular algorithm and discuss shortly some of its properties in section 4. Section 5 then presents the item sharing approach between parsing and generation. On the basis of $\mathcal{UTA}$ and item sharing we then present in section 6 interleaving of parsing and generation through uniform processing and present in detail the novel incremental self-monitoring strategy. We discuss the new approach by comparison with related work in section 7 and outline future extensions.

2 Formal and Linguistic Background

2.1 A relational view on language

It is widely accepted to consider linguistic objects (i.e., words and phrases) as utterance-meaning associations [Pollard and Sag, 1994]. Thus viewed, a grammar is a formal statement of the relation between utterances of a natural language and representations

of their meanings in some logical or other artifical language, where such representations are usually called *logical forms* [Shieber, 1993].

Adopting the simplified assumption that utterances are represented as strings of words, the relationship can be defined more formally as a binary relation R between objects of two different domains, i.e., $R \subseteq S \times LF$, where S is the domain of strings and LF the domain of logical forms.

Parsing as well as generation can be thought of as a program P that is able to enumerate all possible pairs of R for a given element either from the domain of strings or from the domain of logical forms. More precisely, in the case of parsing P computes $\{lf_i | \langle s, lf_i \rangle \in R, i = 1 \ldots n\}$ and in the case of generation $\{s_i | \langle s_i, lf \rangle \in R, i = 1 \ldots m\}$. Thus, P is just a constructive realization of R, no matter whether P constructs R during parsing only or during generation. Since P can construct R for both domains we call P a *reversible* program and R a *P-reversible* relation, in order to emphasize that P can construct R from both directions.

Clearly, up to now we have only assumed that R is a (recursively) enumerable relation. As usual, we assume that the set S of the well-formed strings of a language is enumerable. For a reversible program P this implies that it can enumerate R also from the set LF. Furthermore, we also assume that at least S has an infinite cardinality, so that R has to be defined by some finite recursive device, i.e., a grammar. If the same grammar is used for defining both sets of R, we call this grammar a *reversible grammar*.

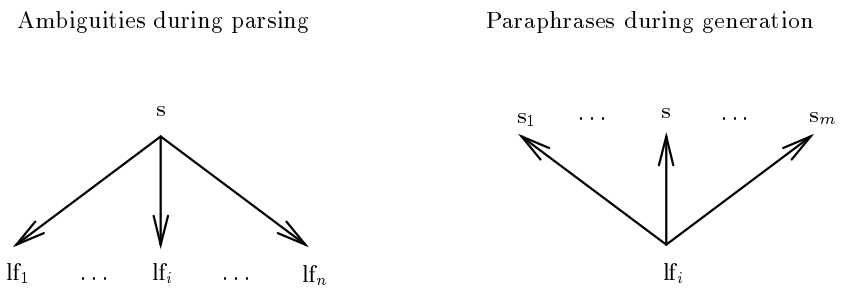

Figure 1: The relationship between ambiguities and paraphrases.

If a sentence s has been associated with more than one interpretation, say $lf_1 \ldots lf_n$, the relation R defined by G will contain pairs $\langle s, lf_1 \rangle \ldots \langle s, lf_n \rangle$ and analogously for a meaning representation lf we will get a set of pairs $\langle s_1, lf \rangle \ldots \langle s_m, lf \rangle$, of all possible sentences that have the same interpretation. Accordingly, the sets are denoted as $R(s)$ or $R(lf)$. The cardinality $\text{card}(R(s))$ of $R(s)$ is defined as the *degree of ambiguity* of s and the cardinality $\text{card}(R(lf))$ of $R(lf)$ as the *degree of paraphrasing* of lf.

Suppose that for some s there exists exactly one semantic expression lf, i.e., $\text{card}(R(s)) = 1$. Then, it is not valid to deduce that if generation is performed starting with lf the resulting set $R(lf)$ is $\{s\}$. However, it is guaranteed that $s \in R(lf)$ (see

also figure 1).

Of course, this kind of "reversibility" is an intrinsic property of each relation. But, if two separate grammars for parsing and generation are used in a natural language system it has to be proven that they describe the same relation; otherwise it would be possible that a sentence which is parse-able cannot be generated and vice versa. Grammar reversibility is very important in practice because it ensures that ambiguous structures and its paraphrases are interrelated. If this is not the case then important aspects of performance like self–monitoring or generation of paraphrases in order to disambiguate ambiguous sentences cannot be modelled properly (see section 6).

Thus viewed, understanding and generation are *dual* processes, in the sense that each sentence which can be understood should also be producible and vice versa. This kind of duality is naturally captured if reversible grammars are used.

2.2 Constraint-logic programming

Since the last decade a family of linguistic theories known under the term *constraint-based grammar theories* play an important role within the field of natural language processing, e.g., LFG[Bresnan, 1982], HPSG[Pollard and Sag, 1994].

In the last few years constraint-based formalisms have undergone a rigorous formal investigation (consider for example [Shieber, 1989; Smolka, 1988; Smolka, 1992]). This has led to a general characterisation of constraint-based formalisms where feature structures are considered to constitute a semantic domain and constraints are considered syntactic representations of such 'semantic structures'. This logical view has several advantages. On the one hand, it has been possible to properly incorporate concepts like disjunction or negation as part of the (syntactic) constraint language and to interpret them relative to a given domain of feature structures (usually defined as graph-like or tree-like structures). On the other hand it has been possible to combine constraint-based formalisms with logic programming, which fits into a new research area known under the term *constraint logic programming* (CLP) [Jaffar and Lassez, 1987].

In constraint logic programs basic components of a problem are stated as constraints (i.e., the structure of the objects in question) and the problem as a whole is represented by putting the various constraints together by means of rules (basically by means of definite clauses). For example the following definite clause specification

$$
\begin{array}{l}
sign(X_0) \leftarrow \\
\quad sign(X_1), \\
\quad sign(X_2), \\
\quad X_0\ syn\ cat\ \doteq\ s, \\
\quad X_1\ syn\ cat\ \doteq\ np, \\
\quad X_2\ syn\ cat\ \doteq\ vp, \\
\quad X_1\ syn\ agr\ \doteq\ X_2\ syn\ agr
\end{array}
$$

expresses that for a linguistic object to be classified as an s phrase it must be composed of an object classified as an np and by an object classified as a vp and the agreement information between np and vp must be the same. All objects that fulfill at least these constraints are members of s objects. Note that there is no ordering presupposed for np and vp as is the case for unification-based formalisms that rely on a context-free backbone, e.g., [Shieber *et al.*, 1983]. If such a restriction is required additional constraints have to be added to the rule, for instance that substrings have to be combined by concatenation.

A general characterisation of CLP is given in [Höhfeld and Smolka, 1988]. Given a constraint language $\mathcal{L}$ and a set $\mathcal{R}$ of relation symbols, $\mathcal{L}$ is extended conservatively to a constraint language $\mathcal{R}(\mathcal{L})$ providing for relational atoms, the propositional connectives, and quantification. In particular, they show how the properties of logic programming carry over to a whole range of constraint-based formalisms, by abstracting away from the actual constraint language in use.

Definite clauses A *definite clause* is an $\mathcal{R}(\mathcal{L})$-constraint of the form:

$$p_1, p_2, \ldots, p_n, \phi \rightarrow q$$

where $n \geq 0$, $p_1, p_2, \ldots p_n$ and q are atoms and ϕ is an $\mathcal{L}$-constraint. We call q the head of a clause and $p_1, p_2, \ldots p_n$ its body. We may write a clause as $q \leftarrow p_1, \ldots, p_n, \phi$ or simply as $q \leftarrow p$. In case the head and the body of a clause are empty, we call the clause an *empty clause*.

A *definite clause specification* is a set of definite clauses. Höhfeld and Smolka show that important properties of conventional logic programs extend to definite clause specifications, especially the existence of a unique minimal model for each interpretation in $\mathcal{L}$.

A *goal* is a possibly empty conjunction of $\mathcal{R}(\mathcal{L})$-atoms and an $\mathcal{L}$-constraint written as $\leftarrow p_1, \ldots, p_n, \phi$ that is, a clause with an empty head (or consequent). An *S-answer* to a goal with respect to a given definite specification S is a satisfiable constraint ψ, such that $\psi \rightarrow p_1, \ldots, p_n, \phi$ is valid for every minimal model of S.

Operational semantics Höhfeld and Smolka provide a generalisation of the SLD-resolution method known from standard logic programming (cf. [Lloyd, 1987]) to definite clauses in $\mathcal{R}(\mathcal{L})$.

The fundamental inference rule for definite clauses in $\mathcal{R}(\mathcal{L})$ is the following *goal reduction* rule (using a slightly different notation from that given in [Höhfeld and Smolka, 1988])

$$p_1, \ldots, p(\vec{x}), \ldots p_n, \phi \Longrightarrow p_1, \ldots, q_1, \ldots, q_m, \ldots, p_n, \rho$$

where $p(\vec{x})$ is the selected element of a goal and

$$p(\vec{x}) \leftarrow q_1, \ldots, q_m, \psi$$

is a variant of a clause of a definite clause specification S and ρ is the result of unifying ϕ and ψ (which we also write as $\text{UNIFY}(\phi, \psi)$).[2]

A proof of a goal g for a clause specification S is a sequence of goals $G, G_1, \ldots$ where each goal G_{i+1} is derived from G_i by applying goal reduction using a variant of a clause of S and the last goal is the empty clause, where its associated constraint is said to be the *computed S-answer* of the goal g. Höhfeld and Smolka show that answers computed in that way are answers for the goal.

Constraint language The constraint language we will use is based on the definition of [Smolka, 1992]. Smolka provides us with a very expressive constraint language including feature equation, conjunction, disjunction, negation, and existential quantification. For the purpose of this work it suffices to use only a small subset of Smolka's constructions, namely feature equation and conjunction.[3]

We will not give a formal definition of the constraint language here since this has already be done (see [Smolka, 1992; VanNoord, 1993]). Instead we directly make use of the "Prolog-flavoured" matrix representation introduced by Van Noord as a readable notation of $\mathcal{L}$-constraints.[4] For example the following constraints on the variable X_0

$X_0\ f_1\ f_3 \doteq \mathrm{c}$,
$X_0\ f_2 \quad \doteq X_0\ f_1\ f_3$

are represented in matrix notation as follows (the variables X_1 and X_2 are computed during the computation of the basic constraint):

$$(1) \qquad X_0 \left[\begin{array}{ll} f_1 & X_1, X_2 \left[f_3\ \ c\right] \\ f_2 & c \end{array}\right]$$

If variables occur only once in a matrix they are omitted. Furthermore, empty feature structures will not been shown explicitly.

The feature structure encoding of the following list

[2] Note, that we directly make use of the so called *optimised goal reduction rule* proven by [Höhfeld and Smolka, 1988] for the general case.

[3] Although we use only simple constructions in order to highlight the new results in a clean but simple way, the generalisation of Höhfeld and Smolka's scheme guarantees that the results of this thesis also carry over to more complex constraint languages. Note further that the same subset has also been used by [VanNoord, 1993] (for the same reasons).

[4] The only important thing to note here, that constraints are based on disjoint sets of variables, constants, and features, as well as descriptor equations, where a descriptor is a (possible empty) sequence of features starting with a variable or a constant. The semantics of $\mathcal{L}$-constraints is defined with respect to the domain of feature graphs.

(2)
$$\begin{bmatrix} \texttt{first}\ a \\ \texttt{rest} \begin{bmatrix} \texttt{first}\ b \\ \texttt{rest} \begin{bmatrix} \texttt{first}\ c \\ \texttt{rest}\ \ end \end{bmatrix} \end{bmatrix} \begin{bmatrix} \texttt{first}\ a \\ \texttt{rest} \begin{bmatrix} \texttt{first}\ b \\ \texttt{rest} \begin{bmatrix} \texttt{first}\ c \\ \texttt{rest}\ \ end \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

will be written more readable using angled brackets as $\langle a\ b\ c \rangle$. The empty list then will be written as $\langle\ \rangle$.

We will also make use of the head/tail representation of lists known from Prolog. Thus to explicitly represent the first element of a list from the rest we write $\langle First|Rest \rangle$ (e.g., $\langle a,\ b,\ c \rangle$ can also be written as $\langle a|\langle b,\ c \rangle \rangle$). Using this notation the difference list of the feature structure

(3)
$$\begin{bmatrix} \texttt{dl} \begin{bmatrix} \texttt{first}\ a \\ \texttt{rest} \begin{bmatrix} \texttt{first}\ b \\ \texttt{rest} \begin{bmatrix} \texttt{first}\ c \\ \texttt{rest}\ \ X \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \texttt{el}\ X \end{bmatrix}$$

will be written as $\langle a\ b\ c|X \rangle$-$X$, and the empty difference list as X-X .

2.3 Specification of grammatical knowledge in $\mathcal{R}(\mathcal{L})$

A grammar G is specified as a definite clause specification where the literals of each definite clause are unary relational atoms.[5] The general form of a *grammar rule* is as follows:

$$p(x_0) \leftarrow q_1(x_1) \ldots q_n(x_n),\ \phi$$

Using our readable notation, a rule can also be represented as

$$p(fs_0) \leftarrow q_1(fs_1) \ldots q_n(fs_n)$$

where fs_i is the feature structure representation of the corresponding variable x_i.

Lexical entries are represented as unit clauses, and grammar rules as non-unit clauses (defining non empty productions) as well as unit clauses (defining empty productions). Lexical entries and empty productions are distinguished using the boolean feature LEX.

Relational atoms are assumed to denote possible constituents of a grammar, either specifically (using for each possible constituent a specific symbol, like *np, vp, pp*) or schematically by only using one symbol, e.g., *sign*. For example, the rule

[5] Considering only unary atoms is not a general restriction since by means of reification we can also express an n-ary atom $r(\vec{X})$ in terms of constraints of a unary relation s(Y) using for example the features REL and ARG_i such that the relational symbol r is viewed as a constant bound to the feature REL and each variable x_i is bound to the corresponding feature ARG_i. Thus $r(\vec{X})$ would be represented as follows: $s(Y), Y\ rel \doteq r, Y arg_i \doteq X_i$.

$$sign(fs_1) \leftarrow sign(fs_2),\ sign(fs_3)$$

expresses that a phrase is built from two phrases, no matter what they are (as long as we do not consider the feature structure). Although the last rule seems to be useless, since it does not say very much about the actual structure of an object, this kind of schematic rule is very prominent in *lexicalized grammars*, since they allow the specification of general combinatory rules, which are independent from individual words.

2.4 Parsing and generation under a CLP view

Considered under the CLP view, the parsing and generation problem consists of a goal that has to be resolved with respect to a given grammar G, specified as a definite clause specification. Parsing and generation differ with respect to the constraints specified for the goal. Since for parsing we want to find the corresponding semantic expressions to a particular string, we require that the constraints at least entail the representation of the string in question, and analogously for generation we require that the semantic expression for which possible strings should be computed is specified. For parsing the feature that represents the string can be considered as an *input variable* and the feature that represents the semantics found can be considered as the *output variable*, and vice versa for generation. We will call the feature that represents the input the *essential feature*, short Ef. For parsing we will assume that Ef is the path ⟨PHON DL⟩ and for generation it is ⟨SEM⟩.

A parsing goal then can be defined as a goal of which the essential feature is ⟨PHON DL⟩ and whose value is bound to the string in question.

Thus, the parsing problem for the string "heute erzählt Peter Lügen" ("today, Peter tells a lie") would be

$$sign\left(\left[\texttt{phon}\quad \langle heute, erzählt, Peter, lügen\rangle\text{-}\langle\ \rangle\ \right]\right)$$

and analogously we define a generation goal as a goal of which the essential feature is ⟨SEM⟩ and whose value is bound to the semantic expression in question. For example for the logical form "heute(erzählen(Peter,Lügen))" ("today(to_tell(Peter,lie)") would be

$$sign\left(\left[\texttt{sem}\left[\begin{array}{l}\texttt{mod}\ heute \\ \texttt{arg}\left[\begin{array}{l}\texttt{pred}\ erzählen \\ \texttt{arg1}\left[\texttt{pred}\ Peter\right] \\ \texttt{arg2}\left[\texttt{pred}\ Lügen\right]\end{array}\right]\end{array}\right]\right]\right)$$

Note that in both cases further constraints may be added to restrict the possible feature structures of found results, for example to be of a specific category, or that the

subcategorization list should be empty. Moreover, it would also be possible to specify the entire syntactic information, for example in the case of generation, to perform some grammar checking. However, what we at least require for parsing and generation is that the value of the essential feature is instantiated.

Restricted parsing problem So far, we only have required that the value of the essential feature should be instantiated. More precisely, we want our algorithm to enumerate all possible feature structures that have a *compatible* value for the value of essential feature. Thus if we want to parse a string, we want the feature structure of that string and analogously for generation we want a feature structure of the input semantics.

Van Noord [1993] has generalized this notation under the term *p-parsing problem*, where parsing in this sense is the general notation for parsing of a string and generation of a semantic expression. More formally, the p-parsing problem consists of a grammar G and a goal q such that $\leftarrow q(X), \phi$.

An answer to a p-parsing problem is a solved constraint ψ such that

- ψ is an answer q with respect to G; and
- $[\![(\phi/Xp)]\!]^{\mathcal{I}} = [\![(\psi/Xp)]\!]^{\mathcal{I}}$

(where $[\![(\phi/Xp)]\!]^{\mathcal{I}}$ is the subgraph found under the path p). In our terminology the path p corresponds to the essential feature Ef. Thus we also use the term *Ef-proof problem* to indicate that parsing and generation are proofs of goals in which the value of the essential feature is instantiated.

3 $\mathcal{UTA}$–A new Uniform Tabular Algorithm

We are now in a position to describe $\mathcal{UTA}$—a new uniform tabular algorithm for parsing and generation with constraint-based grammars. $\mathcal{UTA}$'s basic use is for parsing and generation of grammatical structures. Beside this more traditional use of $\mathcal{UTA}$ its whole new power emerges when it is used in such high-level processing strategies which are based on a tight interaction or interleaving of parsing and generation. Thus, it is important to describe $\mathcal{UTA}$ on a serious level of detail. We do this along the following line:

- data-driven selection function
- uniform indexing mechanism
- agenda-based control
- item sharing between parsing and generation

The first idea is to use the same set of inference rules for parsing and generation – basically we use the Earley deduction proof procedure as introduced in [Pereira and Warren, 1983] – but to use a *data-driven selection function*, so that the element to process next is determined on the basis of the current portion of the input (a string or semantic expression). This enables us, for example, to obtain a left to right control regime in the case of parsing and a semantic head driven regime in the case of generation when processing the same grammar by means of the same underlying algorithm.

Secondly, a new *uniform indexing mechanism* for the retrieval of already completed subgoals (i.e., lemmas) is presented, which can be parameterised with respect to the information used for indexing lemmas. More precisely, in the case of parsing, lemmas are indexed using string information and in the case of generation semantic information is used to access lemmas. Using this mechanism we can benefit from a table-driven view of generation, similar to that of parsing. For example, using a semantics-oriented indexing mechanism during generation massive redundancies are avoided, because once a phrase is generated, we are able to use it in a variety of places.

The uniform tabular algorithm is embedded into an agenda control mechanism, such that new lemmas are first inserted into an agenda. Since lemmas are added to the table according to their priority, we can easily model depth-first, breadth-first and even preference-based strategies. This is even possible in the case of interleaved parsing and generation.

Based on the uniform indexing mechanism we present a novel method of grammatical processing which we call the *item sharing* method. The basic idea here is that partial results computed during one direction (e.g., parsing) are automatically made available for the other direction (e.g., generation), too. Since now items are shared by both directions we call them *shared items*. We show how $\mathcal{UTA}$ is easily extended to make use of shared items.

3.1 Data-driven selection function

The discussion of current approaches for parsing and generation can be summarised as follows: parsing and generation, to be goal-directed, differ basically with respect to the order in which the literals of the body of a clause are selected. For parsing, for example, [Shieber, 1988; Gerdemann, 1991] have used the leftmost selection strategy, where for generation [Shieber *et al.*, 1990; Gerdemann, 1991] use the semantic-head first selection function. The latter should be seen more precisely as a "preference-based" selection function, since in the case a rule has no semantic head, the leftmost element is chosen, or if two elements share the semantics with the mother node, the left one is selected.

However, it is very easy to combine these different strategies used in parsing and generation, such that the selection function expresses a preference for goals with their essential features instantiated (see section 2.4). If we abstract away from a concrete essential feature by assuming that Ef is a variable, then we can define this selection function more formally as follows:

$$\textsc{sf}(q \leftarrow p_1, p_2, \ldots, p_i \ldots p_n, Ef) = \begin{cases} i & p_i, \text{ the first element} \\ & \quad \text{whose Ef is instantiated} \\ 1 & \text{otherwise} \end{cases}$$

In order to use this selection function for parsing or generation we have to specify a path that defines the essential feature (i.e., the phonological or semantic path). Since, the value of this feature will be a string or semantic expression, this means that the selection function prefers those goals which are instantiated with a string or semantic expression. However, now, the grammar itself will be an important source of control, since it defines how complex structure are compositionally crested. For example, if the phonological information is expressed as difference lists and partial strings are combined by string concatenation then the selection function SF "realises" a leftmost strategy. Similarly, if all rules define a semantic head relation SF simulates the semantic head first relation. These can both be true at the same time.

3.2 Uniform indexing mechanism

The purpose of the indexing mechanism employed by $\mathcal{UTA}$ is threefold:

1. avoiding of redundant recomputation by memoing clauses just analysed (i.e., parsed or generated)

2. splitting derived clause into equivalence classes so that necessary lookup operations are restricted only to an identifiable subset

3. using the same mechanism for both parsing and generation

The idea of memoing derived clauses as well as defining equivalence classes for restricting lookup of possible candidates is not a new one (cf. [Earley, 1970; Pereira and Warren, 1983; Kay, 1986]) although with primarily emphasize on parsing. However, considering memoization under a strict uniform and interleaved perspective as followed in this paper has not been described in the literature, to the best of my knowledge.[6]

For parsing, particular data structures have been developed for achieving efficient processing, most notably the *chart* developed by [Kay, 1986] and the *item set* notation developed by [Earley, 1970]. In both approaches the endpoints of a derived string are explicitly used for indexing stored phrases. Unfortunately, we cannot use these well-known approaches for generation directly, because the string is the output of a generator, not the input, of course. For generation, once a phrase has been constructed, we want be able to use it at various places.

[6]Martin Kay (p.c.) currently also investigates uniform indexing mechanisms for parsing and generation, but not under an interleaved perspective.

We now present an indexing mechanism that can be used in the same manner for both parsing and generation. However, since we use the value of the *essential feature* for determining the "content" of internal item sets, the item sets are ordered according to the actual structure of the input. Note that only the selection function and this indexing mechanism have to be parameterised. However, since the only parameter is a certain feature and its value we have achieved *a maximal degree of uniformity for parsing and generation under a task-oriented view.*

The structure of items $\mathcal{UTA}$'s indexing mechanism is based on two data structures, viz *item* and *item set*. An item records the current state of a derived clause. We have to distinguish a clause whose body is not empty from one whose body is empty. The latter will be called *passive clause* and the former *active clause*. In the same sense we distinguish *passive item* from *active item*. An *active item* is a triple of the form:

$$\langle h \leftarrow b_0 \ldots b_n;\ i;\ idx \rangle$$

where $h \leftarrow b_0 \ldots b_n$ is an active clause, i $(0 \leq i \leq n)$ is the index of the selected element in the body of the active clause, and idx is the value of the essential feature of the selected element. The selected element is determined by applying the selection function SF to the active clause.[7]

The general structure of a *passive item* is a triple of the form

$$\langle h;\ \epsilon;\ idx \rangle$$

where h is a passive clause, and idx the value of the essential feature of the head h. ϵ indicates that since the body is empty no selected element can be determined, and hence the selection function should not be applied.

For the representation of the *start item* (i.e., from which processing of a parsing or generation query q starts) we specify the goal statement q as the negative literal of an $\mathcal{R}(\mathcal{L})$-atom that does not belong to the grammar or the lexicon. Thus the structure of the start item is as follows (because q is the only element of the body its index is 0):

$$\langle ans(fs_s) \leftarrow q(fs_s);\ 0;\ fs_s/Ef \rangle$$

Thus, the index is either the string or semantic input in question. Note, that the constraints fs_s of q are shared between q and ans. Hence, the structure of a *goal item* is as follows:

$$\langle ans(fs_g);\ \epsilon;\ fs_g/Ef \rangle$$

i.e., the goal item's clause is passive. Because of the Ef-proof problem it yields that $[\![(fs_s/Ef)]\!]^{\mathcal{I}} = [\![(fs_g/Ef)]\!]^{\mathcal{I}}$. Consequently, the start item and the goal items have the same index.

[7] As long as no misunderstandings are possible, we will use the terms "selected element" and "index of selected element" in the same sense.

The structure of item sets The basic idea is to split the generated items into equivalence classes and to connect these classes, so that each item can directly be restricted to those items that belong to a particular equivalence class. We will call each equivalence class an *item set*. The whole state set then consists of a set of item sets, which we will call a *chart*.

We will use the index idx of an item as index for an item set. Note, that idx equals the value of essential value of selected element of the item's clause (abbreviated as VEF)).

We then require that for each item L in an item set I with index Idx, that VEF(L,Ef) must be the same as Idx. Remember, that for passive items, the essential feature's value is determined from the feature structure of the head element and for active items the essential feature's value is determined by the selected element.

More formally, we can define an item set I as a tuple $\langle AL, PL, Idx \rangle$, where PL is a finite set of passive items and AL a finite set of active items such that:

$$\forall\, pl_i, pl_j \in PL : \textsc{vef}(pl_i, Ef) \;=\; \textsc{vef}(pl_j, Ef) \;=\; Idx \text{ and}$$

$$\forall\, al_i, al_j \in AL : \textsc{vef}(\textsc{sel}(al_i), Ef) \;=\; \textsc{vef}(\textsc{sel}(al_j), Ef) \;=\; Idx$$

Thus all items in one item set share one common property, namely that they are compatible with respect to the value of the essential feature of one of their literals, which is the head in the case of an passive clause, and the selected element in the case of an active clause.

In this sense, an item set can be viewed as a kind of *meeting place* of active and passive items, such that an active item looks for some passive item to resolve with, and vice versa, that a passive item looks for an active item which it can resolve. However, both are identical with respect to the value of their essential feature. If the result of the reduction operation is a new item, this item will eventually be placed in another item set.

It is important to note, that the different item sets are implicitly structured according to the structure of the actual input. For example, if the phonological information is represented as a list, then also the item sets are ordered in a list-like manner. If, on the other side, a tree-like semantic representation is used in the grammar then the structure of the item sets also bears a tree-like structure. This is due to the fact, that the value of the essential feature is used for defining the indexes of the item sets.

3.3 Inference-rules

The control logic of $\mathcal{UTA}$ is a generalisation of the Earley deduction scheme as introduced by [Pereira and Warren, 1983] (see also [Pereira and Shieber, 1987]). It is similar to the one defined by [Shieber, 1988] with the notable distinction that we use a dynamic selection function (where Shieber only uses the left-to-right selection function for both parsing and generation) and that we use a fairly uniform indexing mechanism

(where Shieber only uses indexing efficiently for the case of parsing because his indexing scheme is explicitly based on string positions). Furthermore our approach is the first that makes use of *shared items* between parsing and generation (see section 5).

$\mathcal{UTA}$ operates on two sets of definite clauses, called the *grammar* and the *chart*. The grammar just represents the grammar rules and lexical entries and remains fixed. They are kept in two different data bases (called *Rules* and *Lex* respectively) for supporting efficient retrieval. The chart on the other hand, will be continually augmented with new derived clauses, i.e. lemmas. Whenever a new active lemma is added to one of the chart's item sets, one of its negative literals is selected by calling the selection function SF, i.e., a selected element is determined *on-line*.

Following [Pereira and Warren, 1983] we make use of the following inference rules: *prediction* and *completion*. *Prediction* is used to predict instantiations of grammar rules. Completion will be performed by three inference rules, namely *passive completion*, *active completion*, and *scanning*. In all three cases, passive clauses will be used to reduce appropriate active clauses, where the scanning rule can be seen as a special active completion rule in the sense, that it looks for passive clauses of the lexicon which it uses to reduce the active clause in question.

Using the uniform indexing technique the inference rules can be described more formally as follows. Note, that in each case a new item Ni is deduced by an inference rule two additional things will happen. First, a new empty item set I with the index of Ni will be created in case it does not already exist. This means, that item sets are created *on-line*. Second, Ni is not added to I, but to the agenda *Agenda* according to some determined priority using the function PRIO. This means that a newly created item set I remains empty until the agenda mechanism had chosen an item for insertion into I.

Prediction Let $\langle h \leftarrow b_0 \dots b_n;\ i;\ idx \rangle$ be an active item Ai. Then

prediction(Ai) is:

For every rule $R \in Rules$:
if $\Phi = \text{UNIFY}(\text{ABSTRACT}(\text{SEL}(Ai)), \text{HEAD}(R))$ **and** $\Phi \neq fail$ **then**
with new lemma $Nl = \Phi[R]$ **do**
if $\text{BODY}(Nl) \neq \epsilon$ **then**
make new active item with $Selem = \text{SF}(Nl, Ef)$:
$Ni = \langle Nl;\ Selem;\ Selem/Ef \rangle$
else
make new passive item:
$Ni = \langle Nl;\ \epsilon;\ \text{HEAD}(Nl)/Ef \rangle$
fi;
create item set $I_{\text{INDEX}(Ni)}$ if it does not exist;
ADD-TASK-TO-AGENDA(Ni,PRIO(Ni),$Agenda$)
od.

Here, the selected element of Ai and the rule's head element (the left-hand-side element) are unified and only if unification successed a new item will be created. Thus, prediction deduces a new item on the basis of an instantiated rule. As known from the work of [Shieber, 1985] prediction can lead to arbitrary numbers of consequents through repeated application when used with a grammar with an infinite structured nonterminal domain. In order to avoid such problems, prediction should be performed with an abstraction of the selected element's constraints (which is determined by the function ABSTRACT).[8] Depending on the status of the new item's clause (whether it is empty or not) it is placed in a different item set.

Scanning Let $\langle h \leftarrow b_0 \dots b_n;\ i;\ idx \rangle$ be an active item Ai. Then

$scanning(Ai)$ is:

For every lexical entry $L \in Lex$:
if $\Phi = \text{UNIFY}(\text{SEL}(Ai), \text{HEAD}(L))$ **and** $\Phi \neq fail$ **then**
with reduced lemma $Rl = \Phi[Ai - \text{SEL}(Ai)]$ **do**
if $\text{BODY}(Rl) \neq \epsilon$ **then**
make new active item with $Selem = \text{SF}(Rl, Ef)$:
$Ni = \langle Rl;\ Selem;\ Selem/Ef \rangle$
else
make new passive item:
$Ni = \langle Rl;\ \epsilon;\ \text{HEAD}(Rl)/Ef) \rangle$
fi;
create item set $I_{\text{INDEX}(Ni)}$ if it does not exist;
ADD-TASK-TO-AGENDA(Ni,PRIO(Ni),$Agenda$)
od.

Thus, if a lexical entry can be unified with the selected element of the active item Ai then a new clause is constructed by deleting the unified element from the body of Ai's clause. Following [Pereira and Warren, 1983] we call this operation *reduction*.

The same will be performed for the two remaining completion rules, passive completion and active completion. Thus, by successive application of the completions rules an active item can be transformed to a passive item for which reduction will no more be possible.

Passive-completion Let $\langle h;\ \epsilon;\ idx \rangle$ be an passive item Pi. Then

[8] Here, we follow an approach similar to the one described in [Johnson and Dörre, 1995], by generalising the value of only a small predefined set of constraints, namely those which are known to cause termination problems. The advantage of our approach is that we are able to perform prediction with as many constraints as possible from the selected element. In the parsing literature abstraction has been introduced under the term restriction. More and detailed information on the definition and use of a abstraction/restriction function during parsing see e.g., [Shieber, 1985], [Haas, 1989], and [Samuelsson, 1994]

p-completion(Pi) is:

For every active item $Ai \in I_{idx}$:
if $\Phi = \text{UNIFY}(\text{SEL}(Ai), h)$ **and** $\Phi \neq fail$ **then**
with reduced lemma $Rl = \Phi[Ai - \text{SEL}(Ai)]$ **do**
if $\text{BODY}(Rl) \neq \epsilon$ **then**
make new active item with $Selem = \text{SF}(Rl, Ef)$:
$Ni = \langle Rl;\ Selem;\ Selem/Ef \rangle$
else
make new passive item:
$Ni = \langle Rl;\ \epsilon;\ \text{HEAD}(Rl)/Ef \rangle$
fi;
create item set $I_{\text{INDEX}(Ni)}$ if it does not exist;
ADD-TASK-TO-AGENDA(Ni,PRIO(Ni),$Agenda$)
od.

Thus, passive completion is only applied to active items which are in the same item set as the passive item. In this sense, item sets are meeting places for resolvable active and passive items. Finally, the definition of active completion is

Active-completion Let $\langle h \leftarrow b_0 \ldots b_n;\ i;\ idx \rangle$ be an active item Ai. Then

a-completion(Ai) is:

For every passive item $Pi \in I_{idx}$:
if $\Phi = \text{UNIFY}(\text{SEL}(Ai),\text{HEAD}(\text{CLAUSE}(Pi)))$ **and** $\Phi \neq fail$ **then**
with reduced lemma $Rl = \Phi[Ai - \text{SEL}(Ai)]$ **do**
if $\text{BODY}(Rl) \neq \epsilon$ **then**
make new active item with $Selem = \text{SF}(Rl, Ef)$:
$Ni = \langle Rl;\ Selem;\ Selem/Ef \rangle$
else
make new passive item:
$Ni = \langle Rl;\ \epsilon;\ \text{HEAD}(Rl)/Ef \rangle$
fi;
create item set $I_{\text{INDEX}(Ni)}$ if it does not exist;
ADD-TASK-TO-AGENDA(Ni,PRIO(Ni),$Agenda$)
od.

3.4 Agenda-based control

The inference rules will be embedded in an agenda-based control regime along the line of [Shieber, 1988]. An agenda consists of a list of *tasks* and a policy for managing it. A task is simply an item. Whenever an inference rule creates a new item it is added as a new task to the agenda and sorted according to the given priority function PRIO. If the agenda chooses a task as the next item, and this item is not blocked, i.e., it is not

already a member of the designated item set, it is added to this item set. If we name the agenda mechanism *process* and the query to prove G then

process(G, Ef) is:
make start item Si using G;
ADD-TASK-TO-AGENDA(Si,PRIO(Si),$Agenda$);
while NOT(EMPTY-AGENDA-P($Agenda$)) **do**
let current task Ct = GET-HIGHEST-PRIO-TASK($Agenda$);
if ADD-ITEM(Ct) **then do**
if Ct is a goal item **then**
add Ct to result list Res **fi**
APPLY-TASK(Ct) **od;od**;
if $Res = \emptyset$ **then** return rejection
else return Res
fi.

where *add-item*($Item$) is:
if $\neg\exists I \in I_{\text{INDEX}(Item)}$: I subsumes $Item$
then add $Item$ to $I_{\text{INDEX}(Item)}$
fi.

and *apply-task*($Item$) is:
if $Item$ is passive **then**
P-COMPLETION($Item$) **else**
A-COMPLETION($Item$) **else do**
PREDICTION($Item$);
SCANNING($Item$)
od.

Note, that the selected current task Ct is removed from the agenda and that the inference rules can create new items from it which are then inserted as new tasks to the agenda. However, only a selected task will eventually be added to the chart. Thus, the agenda serves as an temporary storage for new items before they are inserted into the chart. The way, the agenda sorts new tasks depends on the priority assigned to each newly created item. Hence, the priority function determines the search strategy, e.g., depth-first, breadth-first or even best-first.

The function ADD-ITEM performs insertion of an item into the chart. The appropriate item set is selected using the index of the new item. However, before the new item is added to that item set, it is checked whether there exists already an element in that item set which subsumes the new item. This test is known as the *blocking* test [Pereira and Warren, 1983]. Although, we currently use the expensive subsumption operation for performing this test, our uniform indexing mechanism makes it possible to apply subsumption only on a small subset of all possible items already in the chart. Additionally, the agenda mechanism also selects only those items which are currently considered

as relevant for one proof. This is important not only if we follow a best-first search strategy but in particular when we are going to interleave parsing and generation.

The inference rules are called inside the function APPLY-TASK. If the current task (or item) is passive then passive completion is applied. Otherwise active completion is called. The reason why we only consider prediction and scanning if active completion returns false (i.e., creates no new items) is that if active completion is successful this means that for the selected element of the current active item there already exists a derived phrase (made for the same substring or partial semantics) and hence, prediction and scanning would be redundant.[9]

3.5 Parsing and generation with $\mathcal{UTA}$

In order to run $\mathcal{UTA}$ for parsing or generation we only need to specify a query q which contains the input and the value of the essential feature Ef, i.e., the path to the input string or the semantics. For parsing we choose the feature $\langle$PHON DL$\rangle$ and for generation we choose the path $\langle$SEM$\rangle$.

Then $parse(q)$ is:
PROCESS$(q, \langle$PHON DL$\rangle)$.

and $generate(q)$ is:
PROCESS$(q, \langle$SEM$\rangle)$.

For the examples given in section 2.4 the call of PROCESS for parsing looks like

$$\text{PROCESS}(sign\left(\left[\texttt{phon}\quad \langle heute, erzählt, Peter, Lügen\rangle\text{-}\langle\ \rangle\ \right]\right),\ \langle heute, erzählt, Peter, Lügen\rangle)$$

and for generation it looks like

$$\text{PROCESS}(sign\left(\left[\texttt{sem}\left[\texttt{mod}\ heute,\ \texttt{arg}\left[\texttt{pred}\ erzählen,\ \texttt{arg1}\left[\texttt{pred}\ Peter\right],\ \texttt{arg2}\left[\texttt{pred}\ Lügen\right]\right]\right]\right]\right),\left[\texttt{mod}\ heute,\ \texttt{arg}\left[\texttt{pred}\ erzählen,\ \texttt{arg1}\left[\texttt{pred}\ Peter\right],\ \texttt{arg2}\left[\texttt{pred}\ Lügen\right]\right]\right])$$

If we assume a grammar capable of processing these examples where strings are represented through concatenation and semantic expressions through predicate/argument

[9] We have not explicitly required that scanning should only be performed on terminal elements, i.e., active items, whose selected element belongs to a terminal category. The reason is, that in general constraint-based grammars are under-specified in this respect. Of course, if a grammar explicitly distinguishes between nonterminal and terminal elements (as it is the case for instance in LFG), we can easily restrict the application of the scanning rule to terminal elements and the prediction rule to nonterminal elements.

trees then the indices of the created item sets are for parsing (specified in the order they are created during processing):

$\langle heute, erzählt, Peter, Lügen\rangle$; $\langle erzählt, Peter, Lügen\rangle$; $\langle Peter, Lügen\rangle$; $\langle Lügen\rangle$; $\langle\ \rangle$

and for generation:

$$\left[\begin{array}{l}\texttt{mod}\ heute\\ \texttt{arg}\left[\begin{array}{l}\texttt{pred}\ erzählen\\ \texttt{arg1}\left[\texttt{pred}\ Peter\right]\\ \texttt{arg2}\left[\texttt{pred}\ Lügen\right]\end{array}\right]\end{array}\right]; \left[\begin{array}{l}\texttt{pred}\ erzählen\\ \texttt{arg1}\left[\texttt{pred}\ Peter\right]\\ \texttt{arg2}\left[\texttt{pred}\ Lügen\right]\end{array}\right]; \left[\texttt{pred}\ Peter\right]; \left[\texttt{pred}\ Lügen\right]$$

In Appendix A complete parsing and generation examples can be found.

4 Intermezzo: Some Properties of $\mathcal{UTA}$

$\mathcal{UTA}$ is an straight-forward extension of the optimized general SLD-resolution rule whose correctness is proven in [Höhfeld and Smolka, 1988]. It also inherits this property (see [Neumann, 1994b] for more details).

Since $\mathcal{UTA}$ prefers in each deduction step those clauses whose selected element's Ef is instantiated it has a very strong goal-directed as well as data-oriented behaviour in particular for the case of generation. The only relevant parameter our algorithm has with respect to parsing and generation is the difference in input structures. Thus, we are able to characterise parsing and generation in a fairly balanced way without the loss of efficient properties. Hence, we avoid the complications or restrictions that [Shieber, 1988] and [Gerdemann, 1991] are confronted with, because of their "parsing oriented" view of generation. In [Neumann, 1994b] we also show how $\mathcal{UTA}$ is extended to handle empty heads, which are used to describe verb second constructions in Germanic languages like Dutch and German.

The uniform chart mechanism together with the agenda-based control supports the implementation of methods that goes behind simple parsing and generation as we will demonstrate in the next two sections. In particular the on-line creation supports incremental processing for both parsing and generation, and even for the interleaved approach. In the same spirit, the agenda mechanism supports integration of preference-based strategies, e.g., along the lines of [Barnett, 1994] or [Erbach, 1995].

Summarizing, $\mathcal{UTA}$'s properties allow us to consider parsing and generation as the same uniform process which is capable of efficiently controlling the space of possible constructions in a *task specific data-oriented manner*.

5 Item Sharing between Parsing and Generation

We now present a new method for grammatical processing, namely the use of items produced in one direction (e.g., parsing) directly in the other direction (e.g., generation). We will call this method *item sharing* between parsing and generation. If one assumes that parsing and generation are to be performed in isolation, then such method seems to be an overhead. However, in the next section we demonstrate that a strong interleaving of parsing and generation is a necessary prerequisite for modelling high-level performance strategies.

5.1 The basic idea

Assume that $\mathcal{UTA}$ is in the parsing mode. Then in each case a passive item is computed we automatically make available this item also for the generation mode. Thus, for example, if we are going to generate from the semantics of the parsed input we can directly return the previously computed answer during parsing as a result of the generation mode (i.e., if we only consider one paraphrase). Moreover, if we perform generation using a different semantics as the "parsed" one, but which is identical with respect to some partial semantics structures (e.g., some arguments are semantically identical with respect to the "parsed" semantics), then the generator also can "reuse" these partial results determined through parsing. Clearly, this kind of processing makes only sense if during parsing and generation the same grammar and the same basic processing strategy is used.

The restriction of sharing only passive items is plausible for the following reasons. Note that passive items have no selected element. Assume we are in the parsing mode. Then, by means of the definition of item sets, the appropriate value for the index slot for the generation mode can directly be determined on the basis of the semantic information of the passive item. This guarantees that shared passive items produced during the parsing mode are at their right places when they are used by the generation mode.

On the other hand, for active items, in general the chosen selected elements during parsing and generation will be different, and the essential feature of the other direction will be un-instantiated. Therefore, it would not be possible to determine the right place of an shared active item as it is the case for shared passive items.

On the basis of these observations, the structure of an item sharing approach for $\mathcal{UTA}$ is as follows: We assume that $\mathcal{UTA}$ maintains two different agendas, one for the parsing mode and one for generation. This is no overhead, because it allows us to order the tasks of an agenda using, for instance, different preferences. Since item sets are considered as equivalence classes, that are determined on the basis of the value of the essential feature, we assume that parsing and generation have different item sets. Item sets consist of active and passive items. Now, we require that passive items are shared between the item sets determined during parsing and generation. This means, that

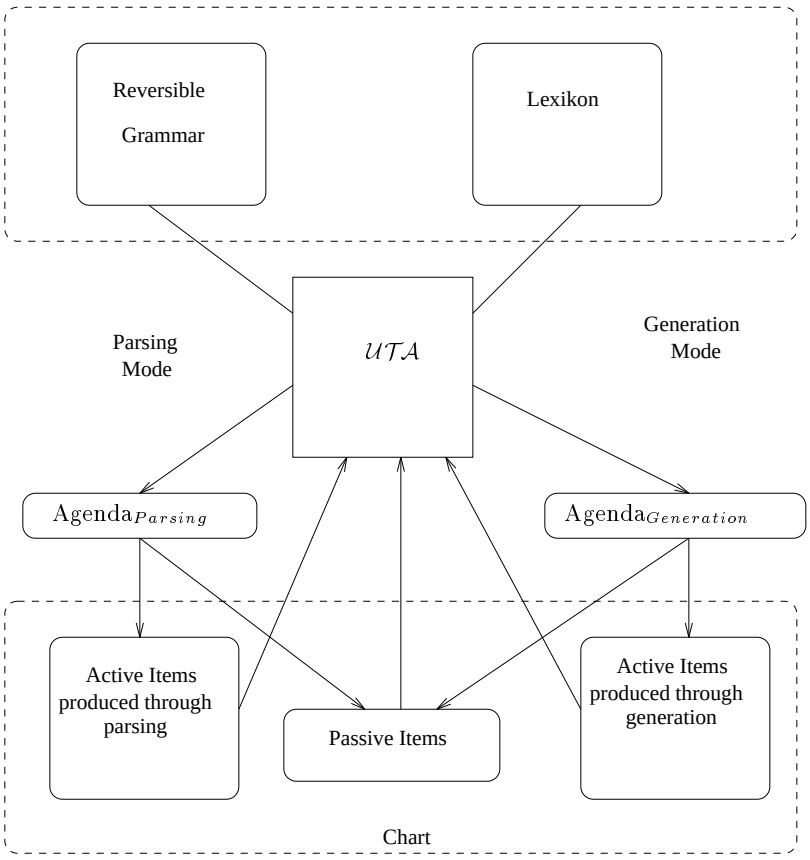

Figure 2: The *item sharing* approach of $\mathcal{UTA}$: During the different modes $\mathcal{UTA}$ maintains different agendas and private active items for the different modes. However, passive items are shared by both directions.

the parser and generator each have there own private active items but can operate on the same set of passive items. Figure 2 illustrates the structure of the item sharing approach.

5.2 Adaptation of the uniform tabular algorithm

In order to adapt $\mathcal{UTA}$ for the item sharing method we extend the structure of an item such that it contains different index slots idx for parsing and generation. Thus, we have

$$\langle L;\ i;\ idx_p;\ idx_g \rangle$$

where L denotes the lemma of an item, i the (position of the) selected element. During parsing the slot idx_p is used and during generation the slot idx_g.

If we are in one of the two possible directions, say parsing, then for active items only the corresponding slot idx_p is filled with the current value of the essential feature. The slot idx_g is unbound which will be denoted by using the symbol *none*. We will use the notation *phon(x)* to denote the value of the essential feature used during parsing and *sem(x)* to denote the value of the essential feature used during generation. Then the general structure of active and passive items is as follows. In the case of parsing, active items are of the form

$$\langle al;\ i;\ phon(i);\ none\rangle$$

and for passive items we have

$$\langle pl;\ \epsilon;\ phon(m);\ sem(m)\rangle$$

where al is an active lemma with selected element at position i in the body of al, $phon(i)$ is the index of the item set al is a member. pl is a passive item with no selected element, and m the pointer to the head of the passive lemma. Note that in the case of passive items, the value of the essential feature for both parsing and generation are determined on the basis of the constraints of the passive lemma's head. This is consistent with respect to the definition of item sets. Analogously, for generation active items are of the form

$$\langle al;\ i;\ none;\ sem(i)\rangle$$

and for passive items we have

$$\langle pl;\ \epsilon;\ phon(m);\ sem(m)\rangle$$

Now the inference rules can easily be adapted to handle such item structures. Firstly, $\mathcal{UTA}$ only considers one index slot, depending on the major mode, for example idx_p for parsing. If a new re-solved lemma (determined through prediction or completion) is active, only idx_p receives the value of the essential feature. The value of the generation slot idx_g is by default *none*. However, if a passive lemma pl is re-solved then also the slot idx_g receives a value determined on the basis of the essential feature specified for this direction (i.e.,the value of the $\langle$SEM$\rangle$ path). This will simultaneously cause the creation of an empty "generation" item set with the corresponding index idx_g. If the agenda mechanism selects pl for insertion into idx_p at some point, then pl is simultaneously and destructively inserted into idx_g, or in other words, pl points into idx_p as well as into idx_g.

If we change the direction mode from parsing and generation and a new passive item pl_g is computed then before pl_g is inserted into the agenda we check whether it is a shared item by applying the blocking test. If this is the case then pl_g is not added to the agenda, since we know that it is already in the chart. This sort of processing is of advantage if we use different preference strategies during parsing and generation since it pretends that shared items will influence the determination of the preference values of next tasks.

5.3 An object-oriented extension of $\mathcal{UTA}$

In order to assign the correct value to the *idx* slot, $\mathcal{UTA}$ has to know in which mode it is. To make this an automatic task $\mathcal{UTA}$ has been embedded in an *object-oriented* environment. In this environment parsing and generation are defined as instances of a class PROOF, and the control mechanism of the underlying object-oriented language automatically will choose the right slot. The structure of the class PROOF is as follows:[10]

```
(DEFCLASS PROOF ()
 (NAME
   RESULT-LIST
   AGENDA
   TASK-COUNTER
   PRIO-FCT))
```

Parsing and generation are simply defined as subclasses of this class and instances are created in the following way:

```
(MAKE-INSTANCE 'PARSE
  :NAME "Parsing direction"
  :AGENDA (MAKE-AGENDA)
  :TASK-COUNTER 0
  :PRIO-FCT #'DEPTH-FIRST)

(MAKE-INSTANCE 'GENERATE
 :NAME "Generation direction"
 :AGENDA (MAKE-AGENDA)
 :TASK-COUNTER 0
 :PRIO-FCT #'DEPTH-FIRST)
```

All functions (with the few exceptions given below) are defined as methods for the class PROOF. This means, that the selection function, the inference rules, the uniform indexing as well as the agenda mechanism are still the same, and implemented only once. In some sense, this means that $\mathcal{UTA}$ only exists one time, but is used by the two different instances. The advantage of using two different instances is that we can easily maintain different agendas or can use specific priority functions for both instances. Thus, our implementation directly mirrors the architecture of the item sharing approach as shown in figure 2.

The only functions that are defined as specific methods for the parsing and generation classes are MAKE-ITEM and ADD-ITEM, and they differ only with respect to one

[10] The object-oriented extension of $\mathcal{UTA}$ has been implemented in CLOS, the Common Lisp Object System [Steele, 1990] (you might want to call $\mathcal{UTA}$ now $\mathcal{UTA}$++). We therefore directly make use of the CLOS-class definition, abbreviated where convenient. In [Keene, 1989] and [Winston and Horn, 1989] good introductions to CLOS can be found.

additional call of a function. For the case of MAKE-ITEM, we have to provide, that if a new lemma is passive, we have to determine values for the slots of the direction that is currently not active. And for the case of ADD-ITEM we additionally have to add the new item to the corresponding item set (which has been created through MAKE-ITEM, if the new lemma is passive) maintained by the inactive instance. Note that this does not mean that the new item is copied, but that the parsing and generation instances actually share this item (internally this means that Lisp provides two pointers to the same internal object).

For illustration of the item sharing approach consider the parsing and generation example given in Appendix A (see also figures 3 and 4). For example, when during parsing the passive item for the partial string "mit Maria" is re-solved (in figure 3 it is the item with task counter 15 and item number 13), then this will cause the creation of an item set for generation with index "mit(Maria)". Additionally a pointer to the passive item 13 is established. In the item sharing approach the structure of the item 13 is:

$$15[pp;\ \epsilon;\ mM;\ m(M)]13$$

Thus if we perform generation with the semantics "sehen(Peter,mit(Maria))" the "parsed" passive item for "mit Maria" with semantics "mit(Maria)" can directly be used during the generation mode.

6 Interleaving of Parsing and Generation

We are now describing how $\mathcal{UTA}$ together with the item sharing method is used for realizing interleaving of parsing and generation. By this we mean that both work together in a fine-grained incremental manner to solve some specific problem. We can distinguish two principle ways of interleaving:

1. during natural language understanding, parsing is supported through interleaved generation

2. during natural language production, generation is supported through interleaved parsing

For example, during parsing of an utterance, generation can already take place for the just parsed parts, by taking into account the parsing results at a very early time of processing. In fact, [Wirén and Rönnquist, 1993] have argued that such a combined view on parsing and generation—in particular following a uniform approach— are worthwhile for exploring highly interactive text-processing facilities such as structure-editing operations, propagation of minimal grammatical changes, or on-line translations, in which the target-language text is generated in parallel with the source-language text [Somers *et al.*, 1990]. Self-control of the parsing process through interleaved generation

is also important for handling under-specified or ill-formed input such that generation is used to "guess" the missing parts or to perform some sort of repair (e.g., to "guess" what the ill-formed utterance probably means). Clearly, additional knowledge-based mechanisms are needed for realizing full functionality, so that interleaved parsing and generation is only one step into that direction —however a substantial one.

During natural language production interleaved parsing is important to obtain hearer-adaptable production of utterances. [Wahlster, 1991] has expressed this under the term *anticipation feedback loop* AFL. The basic idea of the AFL model is the use of the system's natural language understanding part to anticipate the preferred users' interpretation of an utterance which the system plans to realize. In [Jameson and Wahlster, 1982] a local AFL model is used for the generation of elliptical utterances. Here, the basic claim is that anticipation of the way in which the user would be likely to reconstruct a given utterance can help to ensure that the system's utterances are not so brief as to be ambiguous or misleading.

In psycholinguistic research a similar strategy is known under the term *self-monitoring*. Here, there is no deny that people watch over what they are saying and how they are saying it [Berg, 1986]. The basic task of monitoring is to gain information about processing which is not necessarily obvious, i.e., a device is called for which this information can be made available to the speaker or the hearer. It has often been argued in cognitive psychology [Levelt, 1989] that it is highly desirable to find a mechanism that is an integral and independently motivated part of the whole system and one that performs the monitoring function by its own nature. Kempen has noted that "... the addition of a monitor may contribute to the solution of practical and theoretical problems significantly. Take for example the above issue of one-way versus two-way traffic between strategic and tactic components. Suppose the monitor can intercept the linguistic output from the tactical component (preferably before the point of speech) and feed it into a parser/understander. The latter evaluates the generator's utterance from relevant viewpoints and informs (via the monitor) the strategic component of its diagnosis. This would establish the line of communication postulated by Danlos and others without complicating the generator's design — the parser is needed anyway." (cf. [Kempen, 1989], page 15).

In all of the above cited approaches parsing and generation are assumed to work together on a very fine-grained level. In fact, if we can realize interleaving of parsing and generation in such an incremental way the whole natural language system would achieve an important degree of self-control. It is our conviction that system immanent self-control is an important pre-requisite for achieving real flexible and adaptable natural language systems—the core motivation of our uniform work.

6.1 Self-monitoring with reversible grammars

In order to prove the words by actions we are now describing in detail an incremental method for performing self-monitoring and revision during natural language production.

The new approach we are going to present is based on and an improvement of a non-incremental method presented in [Neumann and van Noord, 1992; Neumann and van Noord, 1994]. The basic scientific motivation of this work can be summarized as follows:

> Since during generation the linguistic component is mainly guided by the compositional structure of the semantic input, it cannot determine by itself those particular combinations of partial strings of the whole utterance which will lead to alternative derivations when the hearer is parsing this utterance. This means that possible ambiguities are out of the generator's view, and will only arise during parsing.

For example, the following can happen. A message which is constructed precisely enough to satisfy the conceptual component's goal can be under-specified from the linguistic component's viewpoint. In particular, the generator can *run into the risk of being misunderstood* because of the produced utterance's ambiguity. We call this the **choice problem of paraphrases**.

In order to handle this problem we present in the above cited articles a mechanism which ensures that only non-ambiguous utterances are produced. This mechanism uses the parsing component to monitor the generation component. The relevant communication between the two components is performed using derivation trees. The underlying strategy is based on a comparison of the derivation trees obtained through generation and parsing, where the 'parsed trees' are computed with the output string of the generator. These parsed trees are used as a 'guide' for re-generating the utterance: If the parser yields several readings then each parsed tree is compared from the top downward with the generation tree. For an detected ambiguous subtree the generator is called with the semantics found at the root node of this subtree. The just described mechanism only makes sense for systems in which a single grammar is used for both parsing and generation.

We also described a variant of the monitoring strategy which can be used to paraphrase a given input sentence (for interactive disambiguation) [Neumann and van Noord, 1994]. In this case, the generation component is used to guide the parsing system. Again the proposed technique is possible only in the case of a single, reversible grammar.[11]

6.2 Incremental self-monitoring through uniform processing

The major drawback of the approach mentioned above is that monitoring of a generated string only takes place *after* the whole string has been computed—thus it's degree of

[11] Wojciech Skut (p.c.) has brought to my attention that the self-monitoring strategy has also been very valuable during the writing of an HPSG-style grammar for German since it helped him to test and compare the grammar's degree of ambiguity and over-generation.

interleaving is restricted. However, a fundamental assumption of this non-incremental version is that it is often possible to change an ambiguous utterance *locally* to obtain an unambiguous utterance with the same meaning. Based on this local view it seems plausible to integrate parsing and generation more tightly in the following way: During generation already produced partial strings are parsed to determine the degree of ambiguity. If necessary an ambiguous partial string is revised in order to produce an unambiguous paraphrase of that ambiguous partial string. The successive application of this *incremental generate, parse and revise* technique will end up in an utterance which is unambiguously as possible. Such a strategy works for an example like:

(4) Removing the folder with the system tools
can be very dangerous.

Here, the relevant ambiguity of the whole utterance is forced by the partial string 'Removing the folder with the system tools'. This ambiguity can be solved by restating the partial string, e.g., as 'Removing the folder by means of the system tools' independently from the rest of the string.

However, consider the ambiguous string 'visiting relatives' which can mean 'relatives who are visiting someone' or 'someone is visiting relatives'. If this string is part of the utterance

(5) Visiting relatives can be boring.

then a local disambiguation of 'visiting relatives' is helpful in order to express the meaning of the whole utterance clearly. But if this string is part of the utterance

(6) Visiting relatives are boring.

then it is not necessary to disambiguate 'visiting relatives' because the specific form of the auxiliary forces the first reading 'relatives who are visiting someone'.

This phenomenon is not only restricted on the phrasal level but occurs also on lexical level. For example, 'ball' has at least two meanings, namely 'social assembly for dancing' and 'sphere used in games'. If this word occurs in the utterance

(7) During the ball I danced with a lot of people.

then the preposition 'during' forces the first meaning of 'ball'. Therefore it is not necessary to disambiguate 'ball' locally. But, for the utterance

(8) I know of no better ball.

'ball' cannot be disambiguated by means of grammatical relations of the utterance.

6.2.1 Basic problems of incremental monitoring

The problem is that the monitor must be dynamically configured during incremental processing time of single utterances in order to decide

- when the test of ambiguity should take place and
- which partial strings should be revised?

Technically it is possible to check and revise each partial result of the generator. But, without any control, the monitor would try to disambiguate each local ambiguity; it is hard to imagine that the resulting generator would produce anything at all.

Clearly, an utterance can only be said to be (un)ambiguous with respect to a certain *context*. The assumption is that usually an utterance which is not ambiguous with respect to its context will remain unambiguous if it is part of a larger utterance.

It may be possible to restrict the context during the production of a partial utterance to grammatical properties, e.g. to the information associated with the *head* which selects the phrase dominating this partial utterance. Such an approach can be integrated in head-driven generators of the type described in [Shieber *et al.*, 1990]. For example, assume that for each recursive call to the generator the revised monitor is called with an extra argument `Head` which is to be used as contextual information when the embedded parser is called to test whether the string in question is ambiguous. Thus, suppose we are to generate from the logical form $during'(ball')$.

A head-driven generator first produces the word `during` as the head. Next an NP with logical form $ball'$ has to be generated. For this logical form the generator chooses the word `ball` which is however ambiguous. For this partial utterance the monitor is called, using the head information of `during`. However, being an argument of the head `during`, only one of the readings of `ball` is possible. Therefore, the monitor simply 'confirms' the choice of the generator. Thus, the assumption here is that this ambiguity will be disambiguated later on by combining this string with its head. The main problem of such an approach is that

- either each ambiguous partial string has to be revised immediately or
- revision is delayed until the previous recursive call of the generator has been finished.

In the first case the monitor would also revise irrelevant ambiguities. The latter point would mean that revision can only be performed after the whole utterance has been produced. But then such an incremental method would just simulate the non-incremental method.

6.2.2 A look-back strategy

It seems to be more plausible to test the ambiguity of a partial string with respect to already produced partial strings. Based on this idea the notation of context is considered as follows: The *context* of a partial string α with constituent A is the string β of the adjacent constituent B of A. Parsing is then performed on the "extended" string $\beta\alpha$, to test whether this string leads to some ambiguity. If the "extended string"

is either not parse-able or is not ambiguous we conclude that the newly produced string α does not force ambiguities in the current state of computation of generating the final string.

For example suppose that an utterance with meaning 'Remove the folder by means of the system tools.' has to be produced. Furthermore, suppose that the partial string 'Remove the folder' has been generated using a rule 'vp $\rightarrow$ v, np, pp'. Now, the result of generating the pp is 'with the system tools'. In order to check whether this string is ambiguous 'the folder' is used as context and the string 'the folder with the systems tools' is parsed. This string is parse-able if a rule e.g., 'np $\rightarrow$ np, pp' exists. If it is parse-able then a source of ambiguity has been found, so that pp should be revised. If revision is not possible, then revision of the previous chosen vp should take place. However, if the rule 'vp $\rightarrow$ v, pp, np' had been chosen, and the currently produced string is "the folder", then the extended string to parse would be "with the system tools the folder". In this case, however, the string would not be parse-able. For the monitoring strategy this means, that at this point of computation, no statement of a possible ambiguity can be made, so the revision should not take place. In other words, the newly produced string "the folder" does not cause a relevant ambiguity in the current domain of locality spanned by the vp rule.

The proposed approach realizes a kind of *look-back* strategy, in the sense, that the monitor looks back to already produced substrings, in order to test whether a new string together with previous produced substrings causes ambiguity. For the method described so far, we actually have made a look-back of one adjacent constituent. In principle, however, it is possibly also to take into account the adjacent element of an adjacent element, leading to a look-back(n) strategy. The degree of look-back used, directly influences the *degree of ambiguity* we are going to consider. For example suppose we have the following grammar (s, np, vp are non-terminals, the other symbols are terminal elements):

1. s $\rightarrow$ np vp
2. vp $\rightarrow$ a b c d
3. vp $\rightarrow$ a np
4. np $\rightarrow$ x y
5. np $\rightarrow$ b c d

Assume that we have first chosen the vp rule 2., and the newly generated element is d (we assume a left-to-right scheduling). If we are following a look-back(1) strategy, then we have to parse the string 'cd'. This string, however, is not parse-able, so we do not try to revise it at that point. However, if we would use look-back(2), the string to parse would be 'bcd'. This string is parse-able, so there is the possibility of an ambiguity.

We are now going to describe how the look-back strategy informally described above is integrated with $\mathcal{UTA}$ in order to perform the desired incremental monitoring strategy.

Basically, we have to discuss the following questions:

1. How is the context determined and used for locating potential ambiguities?
2. How do we realize revision within $\mathcal{UTA}$?

The first question is concerned with the problem of determining context. This implies that we have to consider the possible different granulations the shape a context can have. For example, does it make sense to consider only one word as context or should it be better the string of complex constituents? This question will be considered after having introduced how revision should take place, because activation of revision is triggered after having determined a context, but it is possible to discuss revision by just assuming that context has already been determined.

6.2.3 Performing revision within $\mathcal{UTA}$

It turns out that performing revision during generation of an utterance using $\mathcal{UTA}$ is not that difficult as it might be at a first glance.

Recall that $\mathcal{UTA}$ keeps track of partial (complete or incomplete) results using an agenda and a chart. The agenda is used to maintain all newly created items before they are added to the chart. The selection strategy used by the agenda determines in which order the items are added to the chart. Following a depth-first strategy, for instance, then only those items are considered that eventually can contribute to the complete generation of the first possible utterance. All remaining alternative items are only added to the chart in the case that additional paraphrases are requested, e.g., when all possible strings of a given semantic expression shall be computed.

If an item has been added to the chart, the different inference rules are applied which eventually creates new items which are then added to the agenda. But note that only those created items which have been added to the agenda will be considered during further computation. By means of this "built-in" mechanism revision can be performed as follows: Suppose that we have deduced a new passive item p. This means that we have computed a new partial string. If p is added to the chart, by means of passive completion it is checked whether p can reduce an active item a. Then, before a is actually reduced using p it is checked whether p causes an ambiguity using an appropriate context.

Only if no ambiguity can be determined, the reduction of a is performed and the resulting new item is added to the agenda. On the other side, if an ambiguity is recognized, then reduction will not be performed, and as a consequence no new item is created. This implies for a, that reduction of its selected element will only be performed if there is another alternative for p available on the agenda (or items which lead to the computation of the alternative). However, this alternative item will automatically be added to the chart by the agenda at some later point. In some sense, this kind of

processing means that the selected element has implicitly been marked, and the agenda will choose an alternative item which corresponds to a selection of an alternative rule.

If no alternative for p can be deduced (i.e., either no further alternative exists, or no unambiguous alternatives exist), then a will never be completed. However, this means that the agenda automatically will add an alternative item of a (if present) to the chart, which then might be combined with p. Note that this reduction would be performed by active-completion, and hence, would reuse results of previously made computations. If this is the case, the marker of p implicitly has been pushed one level up. Since, the whole process is performed recursively, it might be the case that markers are pushed implicitly up to the initial root node. However, in all cases, we can benefit from the results of previously made computations.

We will use our pp-attachment example at that place to clarify the strategy. We are assuming the following simple grammar:

1. s → np vp
2. vp → v np pp
3. vp → v pp np
4. np → det n
5. np → np pp
6. pp → prep np

We assume that these rules are added to the agenda according to the order in which they are specified in the grammar. Using a depth-first selection strategy for the agenda, rule 2. is processed before 3. At some point the pp is produced, and will be used by passive completion to reduce an instance of rule 2. However, before the pp of vp is reduced, the string of the np is used as context for checking whether the pp causes ambiguity. Therefore, we parse the string of np-pp, and actually detect an ambiguity. For the pp, however, we have no further alternatives available on the agenda, so rule 2. cannot be reduced completely, i.e, for that rule the inference rules cannot create items to put on the agenda. However, the agenda mechanism guarantees that rule 3. will be selected. Reducing rule 3. by means of active-completion will first use the pp for reduction, assumed without ambiguity problems. Next the np should be used for reduction. Before that, however, the string of pp-np is monitored, which however cannot be parsed, and hence no revision is necessary. Thus, rule 3. will be reduced by the np to give a completely reduced vp, which then is used for reduction of rule 1.

Based on the observations made above, we can adapt $\mathcal{UTA}$ for performing revision in the following way (assuming that we already know how to detect ambiguities in the incremental mode, see next subsection): Revision should only take place if there exists a passive item which can be used for reducing an active one. Thus, we only have to consider revision for the completion rules *active completion*, *scanning*, and *passive completion*.

In all three cases we add a further conditional statement around the body of the

for all loop, namely that the body should only be evaluated if revision is not requested. For example, the *passive completion* rule is changed as follows (only the relevant parts are expressed explicitly):

p-completion(Pi) is:
  For every active item $Ai \in I_{idx}$:
   if Φ = UNIFY(SEL(Ai), h) **and** $\Phi \neq fail$ **then**
    if NOT(AND ($Monitor?$,REVISION-P($\Phi[Ai]$,Pi))) **then**
     with reduced lemma $Rl = \Phi[Ai -$ SEL(Ai)] **do**
     ...
    od

In the relevant part of the new code we have added a new condition which says that the next operations (i.e., putting a just reduced active item on the agenda) will only be performed if the monitor mode is switched on (which is done by using a global variable *Monitor?*, whose boolean value indicates whether processing should be performed with or without incremental monitoring) and if no revision has taken place, which is determined the predicate REVISION-P.[12]

In the same manner *active completion* and *scanning* are modified. The definition of *revision-p* is as follows:

revision-p(Ai, Pi) is:
  with $ExtendedString$ = GET-CONTEXT(Ai, Pi,n);
   if $ExtendedString$ **then**
    with $ParsedResult$ = PARSE($ExtendedString$);
    if AND($ParsedResult$,AMBIGUOUS(Ai,$ParsedResult$))
     then true else false fi
   else false fi.

The function GET-CONTEXT determines the contextual information. If so, the parser is called with the extended-string, built inside GET-CONTEXT, using a look-back of n, which value is set globally. To be more precise, first a new string is computed on the basis of the context and the passive item's string, and then the parser is called. Only if the parser successfully obtained one or more readings and if the result is ambiguous should revision take place.

Note that the way $\mathcal{UTA}$ maintains the agenda and the chart, the incremental method "simulates" marking and revision of generated derivation trees as is done explicitly by the non-incremental method. However, marking is done implicitly — it is just a side effect of $\mathcal{UTA}$ by not creating items which could cause ambiguity problems. Furthermore, because monitoring is applied on intermediate results, it is actually performed

[12] Using a globally set flag to trigger incremental monitoring is useful if the flag can be switched off in a kind of *any-time mode*. For example, if the overall system receives important time constraints and if it is possible to change the value of *Monitor?* from true to false interactively, the remaining semantic expression is generated without monitoring. We actually have *implemented* this any-time strategy.

incrementally.

6.2.4 Performing ambiguity checks within $\mathcal{UTA}$

We now turn our attention to the problem of testing whether a new partial produced string causes ambiguity or not. To solve this problem, we have to specify how an appropriate context is determined, how this context is used for parsing, and how the result of parsing is analysed with respect to its ambiguity.

Determination of context The basic assumption behind the use of contextual information during the incremental monitoring strategy is that it only makes sense to test whether a partial string, say α, is ambiguous with respect to a larger string which entails α. Such a larger string will be built by means of concatenation of α and some other already produced string, which we will call the *contextual string* of α.

Since revision will be performed before a passive item Pi is used for reduction of an active item Ai, this active item defines the domain of locality from which contextual information can be determined. Completion will be performed if Pi and the selected element of Ai can be unified. Therefore, only those elements of the body will be considered as possible contextual strings, that have already been deduced as subgoals of the active item.

Note that the call of the incremental monitoring mechanism, i.e., the call of REVISION-P is performed in a completion rule before the new reduced item is computed but after unification of the passive item with the selected element of the active item has been done. This guarantees that monitoring is only performed on consistent structures. As a side effect of unification, the derivation tree of the passive item is unified into the derivation tree of the head of the lemma of the active item.[13]

For example, assume that we have reduced the grammar rule $vp \leftarrow v, np, pp$ up to the point where we only need to complete the pp in order to complete the vp. The corresponding active item would be of form

$$\langle vp \leftarrow pp;\ 0;\ \textsc{vef}(0, Ef) \rangle$$

At that point the derivation tree represented as part of the constraints of vp is (making use of useful abbreviations):

[13] We assume that derivation trees are represented as part of the head's constraints of each rule and lexical element using the feature DERIV. The internal structure of this feature consists of the features LABEL which value is a constant that uniquely identifies this clause, and DTRS which value is a list of the derivation trees of the elements of the body of a clause. Additionally, two features PHON and SEM are used as pointers to the string and semantics of the clause, and are used as an interface for parsing and generation. We are using this representation since completion causes the removal of the completed elements from the body of a clause, so the elements of the body cannot be used directly. Thus, we will determine the contextual string of a passive item on the basis of the derivation tree represented as part of the constraints of a resolved active item.

$$\begin{bmatrix} \texttt{rn} & \textit{vp3} \\ \texttt{phon} & \langle remove, the, folder \rangle\text{-}P \\ \texttt{sem} & \ldots \\ \texttt{dtrs} & \left\langle \begin{bmatrix} \texttt{rn} & \textit{v5} \\ \texttt{phon} & \langle remove \rangle\text{-}P1 \\ \texttt{sem} & \ldots \\ \texttt{dtrs} & \langle\ \rangle \end{bmatrix}, \begin{bmatrix} \texttt{rn} & \textit{np3} \\ \texttt{phon} & \langle the, folder \rangle\text{-}P2 \\ \texttt{sem} & \ldots \\ \texttt{dtrs} & \textit{"its\ tree"} \end{bmatrix}, Tree \right\rangle \end{bmatrix}$$

where the variable $Tree$ is a pointer to the derivation tree of the selected element pp, which is still un-instantiated.

After successful unification of a passive item pp with the selected element, the value of the variable $Tree$ in the above derivation tree is:

$$\begin{bmatrix} \texttt{rn} & \textit{pp1} \\ \texttt{phon} & \langle with, the, tools \rangle\text{-}P \\ \texttt{sem} & \ldots \\ \texttt{dtrs} & \textit{"its\ tree"} \end{bmatrix}$$

Now, we take this representation as the basis for determination of the contextual string of the pp's string "with the tools" making use of a *look-back* strategy as already informally described above.

The value of the DTRS feature is a sequence of the derivation trees of the corresponding elements of the body. In this context, we will call the value of the DTRS feature the *sequence of sisters* of the node represented by the clause's head element.

Since we consider the sister nodes as totally ordered in a sequence, a look-back one strategy (written as look-back(1)) of the selected element, is just the choice of its left or right sister node. Thus, for the example above, we choose the node labelled *np*3. From this derivation tree we choose the value of the STRING feature as contextual string. Since, we assume that strings are represented as difference lists, it will be the case, that the string of the root node of the derivation tree of np already entails the string of *pp*. Thus, we can directly start parsing of this string, to test whether this string is ambiguous.

Note that in the above example we have implicitly assumed, that the elements of the body are processed in a left-to-right manner. Of course, in the case of generation this is not the general case. It might be possible, that for example, the *pp* is completed before the *np* is. In this case, we would have no (left) sister to be use-able as contextual string for the *pp*, because the derivation tree of the *np* still needs to be constructed, which means that the position of this derivation tree within the sequence of sisters is still occupied by an un-instantiated variable. If this is the case, we conclude that for the *pp* no statement about ambiguity can be made, and therefore, no revision should take place. After the *np* has been completed, monitoring for the *np* will eventually take place. But now, there is a choice point for the *np* either to choose its left or right sister as the base of contextual information, or both.

We can directly generalise the informal description of a look-back(1) strategy to a look-back(n) strategy, if we not only consider the left or right sister node of the selected element as context but the sequence of the n left or right sisters of the selected element. In order to do this we have to consider the following cases:

- one of the n sisters is un-instantiated, and
- there are less then n possible sister nodes to the left or right of the selected element.

The first case means that there is a sister which derivation tree has still not been computed. This means that we cannot determine the whole contextual string corresponding to the n sisters, and we conclude that no contextual string exists. The second case means that the whole set of left or right sisters of the selected element can be used as contextual information by actually performing a look-back of less than n. In that case we use the corresponding contextual string spanned by the sisters and use it for the ambiguity check.

For a more readable definition of the look-back(n) strategy, we make use of the notation $subseq(i, j)$, which is a subsequence of elements ranging from i to j. For example, $subseq(3, 5)$ denotes the subsequence $\langle c, d, e\rangle$ of the sequence $\langle a, b, c, d, e, f, g\rangle$. Empty production will be handled so that if the sequence contains the name of an empty production we just skip this element. For example, if a and b are empty productions, then the sequences $\langle a, c, a, b, d, e\rangle$ and $\langle c, d, e\rangle$ are considered as being equal. The notation "the string of $subseq(i, j)$" means the string built by a left to right concatenation of the strings of the elements of the subsequence (modulo empty productions). We will say that a "$subseq(i, j)$" is instantiated if for each element of the subsequence its derivation tree is instantiated. Thus, the look-back(n) strategy can be expressed as follows:
Let $\langle d_1, \ldots, d_m\rangle$ be the sequence of sisters of the derivation tree of a rule and let d_i be the derivation tree of the "unified" selected element of the rule, and α its string. Let ll be the length of $subseq(1, i-1)$ and rl be the length of $subseq(i+1, m)$. If $n > ll$ then let n be ll, and analogously let n be rl, if $n > rl$. Then,

- if $subseq(i-n, i-1)$ is instantiated but not $subseq(i+1, i+n)$ then let β be the string of $subseq(i-n, i-1)$; let $\beta\alpha$ be the extended string;
- if $subseq(i+1, i+n)$ is instantiated but not $subseq(i-n, i-1)$ then let β be the string of $subseq(i+1, i+n)$; let $\alpha\beta$ be the extended string;
- if $subseq(i-n, i-1)$ and $subseq(i+1, i+n)$ are instantiated with strings β and γ respectively then let $\beta\alpha\gamma$ be the extended string;
- otherwise, no contextual string exists, which is indicated by the boolean value **false**.

This definition is used inside the function GET-CONTEXT (which is called inside REVISION-P, see above) which receives as input an active and a passive item and returns either an extended string or false, i.e.,

get-context(Ai, Pi, n) is:
with $Dtrs$ = GET-DTRS(Ai);
with $Lsisters$ = GET-LEFT-SISTERS(Ai,LABEL(Pi));
with $Rsisters$ = GET-RIGHT-SISTERS(Ai,LABEL(Pi));
"apply look-back(n) on lsisters and rsisters;"
if *ExtendedString* **then**
ExtendedString **else false fi**.

We first extract the sisters of the derivation tree of the active item Ai, i.e., the value of the path $\langle deriv, dtrs \rangle$ of the constraints of the active item's lemma's head. We then split this list into a left and right subsequence, where the passive item (which corresponds to the unified selected element of Ai) serves as the splitting point. Next, we apply the look-back(n) strategy, and either return an extended string or **false**, if no such exists.

Check ambiguity Next we call the parser (i.e., we run $\mathcal{UTA}$ in the parsing mode), whose task is to parse the extended string. If the extended string cannot be parsed, we conclude that no revision is necessary, and the call of REVISION-P terminates with **false**. However, if the parser returns one or more results (which corresponds to semantic readings of the extended string), we apply the ambiguity check performed inside the function AMBIGUOUS (see below). Only if a parsed result exists and the result is ambiguous, REVISION-P returns **true** which will cause revision of the new string spanned by the passive item.

The ambiguity check is performed as follows. First we delete all spurious ambiguities, i.e., for a pair of derivation trees which have the same semantics we only retain one.[14] After this operation we may have either only one reading or a set of readings. The latter case means that there are different possibilities to assign a meaning to the extended string, therefore revision for the new string should take place.

The former case is a bit more complicated. Although this case means that the extended string has been analysed as unambiguous (since we have obtained only one result), it might be the case that this reading is the same as that of the semantic expression of the active item's lemma. In this case, we have just detected a spurious ambiguity, and therefore revision should not take place. If on the other side, the semantic expression is not equal to that of the active item, we have found an possible ambiguity, and hence, revision should take place.

[14]The test for spurious ambiguity thus serves as a filter. Clearly, the current formulation of the test might be to simple. However, in principle it is not difficult to exchange it with a more complex test as long as the semantic representation of the grammar would support application of such a more complex test.

The following description of the function AMBIGUOUS summaries the different cases:

ambiguous($ParsedResult, Ai$) is:
with $ReducedResult$ = "delete spurious ambiguities";
if CARD($ReducedResult$) > 1 **then**
true else
if SEM($ReducedResult$) = SEM(Ai) **then**
false else true fi fi.

6.2.5 Using shared items during incremental monitoring

The main advantage of the incremental method using $\mathcal{UTA}$ described so far is that we benefit from the use of the chart during the monitored generation strategy, because also in that case we can reuse previously made computations. Since revision is automatically performed by the agenda mechanism of $\mathcal{UTA}$ (by not creating items for those structures where an ambiguity as been detected), the main effort we have to spend to realize monitoring is the parsing operation performed on extended strings. (The determination of the contextual string is not a time critical operation.) We now show how the incremental monitoring method can be made more efficient by making use of the *item sharing* approach described in 5.

Recall that in the item sharing approach passive items that have been computed in one direction can directly be used in the other. Following the method described in 5 $\mathcal{UTA}$ maintains different agendas, item sets and active items for the parsing and generation mode, but passive items are shared during both directions. The object-oriented realization of the item sharing approach allows the parser (i.e. the parsing mode of the uniform algorithm) to be chart-based even when it is called inside the generator. Thus, if the parser is called via monitoring it can reuse previously self-made results at any stage.

By use of the item sharing approach partial results (i.e., passive items) are continually made available for the other direction. However, this means that for both directions it is the case that one direction can reuse results from the other directions. For example, for the interleaved parsing mode this means that it can reuse results computed through generation when making the ambiguity check. During this job, however, it can provide results for the generator of which the generator can make use. This means, that parsing results are used through generation and generation results are used through parsing in an interleaved mode.

6.3 Properties of the incremental self-monitor

The incremental monitoring method can be seen as an additional restriction to $\mathcal{UTA}$ to keep track only those computed partial results which do not force ambiguities. Note that monitoring is only triggered by the completion rules and will only be performed on

consistent structures. The effect of monitoring is that $\mathcal{UTA}$ will only consider a *subset of possible answers*, namely those which are un-ambiguous. If no un-ambiguous string can be produced then the resulting set of answers is empty. However, if the algorithm finds an answer then it is correct. In this sense the monitor just further constraints the set of computable answers for a given semantic expression.

Degree of resolved ambiguity There are two parameters which influence the behaviour of the incremental monitoring strategy: the concrete value of n for the look-back strategy and the degree of the nodes of a derivation tree, which corresponds with the length of the right hand side of the rules. We will call this the *branching factor* of the grammar. The maximal possible degree of a node will be denoted as *maximal branching factor*, and corresponds to the rule with the largest number of right-hand side elements defined in a grammar.

Recall, that the variable n refers to the number of sisters of a selected element of an active item which have to be considered as context. Suppose we have chosen 1 as the value of n, i.e., we are following a look-back(1) strategy. Furthermore, assume we have two grammars G_1 and G_2 which are weakly equivalent, and where the maximal branching factor of G_1 is 2 and that of G_2 is some integer m greater than 2.

For G_1 a look-back(1) strategy means, that in each case where the incremental monitor mechanism is activated the newly determined extended string is identical with the whole string of the constituent defined by the active item. The reason is that when using a grammar defining binary rules, only when both elements of a rule have been deduced, monitoring will take place; otherwise no contextual information would be available. This implies, that all possible ambiguities will be detected and that if the incremental monitor generates an utterance, then this utterance is unambiguous.

For G_2 a look-back(1) strategy means in general, that only a substring of the string defined by a constituent will be taken into account when building an extended string. But then, it is possible, that not all possible ambiguities will be detected (see also section 6.2.2). Consequently, this means that if the incremental monitor generates a string, this string need not necessarily be unambiguous.

Putting both together, we obtain a different result (wrt. the degree of ambiguity of a "monitored generated string") using the same value of n, but on grammars which only differ with respect to their maximal branching factor. Of course, if we want to make sure that our algorithm behaves in the same way for grammars with different maximal branching factor, i.e., if it is to guarantee that only unambiguous strings are generated, then we have to choose the maximal branching factor of the grammar as the value for n when performing the look-back strategy.

The discussion made above directly reveals the problem of determining the appropriate value for the look-back strategy. If we choose the maximal branching factor, then we obtain unambiguous strings, for the price of high computational effort. On the other side, if we choose a small value for n we reduce the effort but will eventually not

obtain an unambiguous string. Furthermore, it cannot be guaranteed that we actually have considered all relevant ambiguities.

In order to compromise between computational effort and the degree of resolved ambiguities, we have to consider some additional criterions, which are used to decide whether an ambiguity check should be applied to a newly generated string. Assumed we have such criterions they can easily be used during monitoring, such that during the call of GET-CONTEXT this information is used firstly to check whether for the passive item an ambiguity check should take place, and second on each sister "consumed" by the look-back strategy the tests are applied. Only if the passive item and its sisters fulfill the conditions expressed by these criterions an extended string will eventually be delivered. This provides the possibility to restrict the application of the monitoring strategy, for instance, on grammatical information. For example, it would be possible to restrict monitoring only for maximal projections or only for those structures which are known to cause ambiguities (e.g., pp-modifiers, coordinations). In our implementation we have already built in mechanisms that can take into account such additional grammar specific information. However it is a matter of future investigation (primarily on the linguistic side) to achieve meaningful and realistic criterions.

User-driven control and any-time mode The incremental monitoring mechanism has fully been implemented and integrated into $\mathcal{UTA}$. Although we currently do not use preferences-based strategies we have paid attention to be as flexible as possible with respect to this future extension. For example, if $\mathcal{UTA}$'s Prolog-like interactive mode is activated during generation and parsing a user can choose which of the results computed during parsing should be ignored for the ambiguity check. This way, the user can interactively specify which reading she prefers.

Furthermore, it is possible to switch off and on the monitor interactively by just changing the Boolean value of the global variable MONITOR? which is used to trigger monitoring. In our current implementation, this can be done in combination with the Prolog-like interactive mode. If the flag is switched off further generation is automatically continued without monitoring. Using this mechanism it is possible to simulate an *any-time mode* of the incremental method.

Limitations It should be clear that monitoring and revision involves more than the avoidance of ambiguities. [Levelt, 1989] discusses also monitoring on the conceptual level and monitoring with respect to social standards, lexical errors, loudness, precision and others. Obviously, our approach is restricted in the sense that no changes to the input logical form are made. If no alternative string can be generated then the planner has to decide whether to utter the ambiguous structure or to provide an alternative logical form.

During the process of generation of paraphrases it can happen that for some interpretations no unambiguous paraphrases can be produced. Of course, it is possible

to provide the user only with the produced paraphrases. This is reasonable in the case that she can find a good candidate. But if she says e.g., 'none of these' then the paraphrasing algorithm is of no help in this particular situation.

Meteer [1990] makes a strict distinction between processes that can change decisions that operate on intermediate levels of representation (*optimizations*) and others that operate on produced text (*revisions*). Our strategy is an example of revision. Optimizations are useful when changes have to be done during the initial generation process. For example, in [Neumann and Finkler, 1990] an incremental and parallel grammatical component is described that is able to handle under-specified input such that it detects and requests missing but necessary grammatical information.

7 Discussion and Future Extensions

7.1 Related work

$\mathcal{UTA}$ can be seen as extension of Shieber's uniform algorithm. It uses a dynamic selection function (where Shieber only uses the left-most selection function for both direction) and a truly uniform indexing mechanism (where Shieber handles indices efficiently only during parsing). Gerdemann [1991] also presents an extension of Shieber's algorithm that tries to make efficient use of indexing during generation. However, his degree of uniformity is restricted since he actually uses different indexing mechanisms for parsing and generation.

$\mathcal{UTA}$ has a stronger goal-directed behaviour than the semantic head-driven algorithm described in [Shieber *et al.*, 1990], because it uses a semantic-oriented selection for all rules of the grammar (where Shieber et al consider only a subset of the rules; all other rules are processed in a simple left-to-right top-down manner). Furthermore, they do not make use of a chart. Van Noord [1993] has extended this algorithm also for head-corner parsing. A main problem with his approach is that it does not support incremental processing.

The use of the essential feature Ef as the single parameter of $\mathcal{UTA}$ is comparable to Strzalkowski's essential argument approach [Strzalkowski, 1994]. However, he uses this information only off-line during grammar compilation in order to obtain specific parsing and generation grammars.

In [Erbach, 1995] a uniform algorithm based on bottom-up Earley deduction is presented that makes use of a flexible indexing scheme, however only for the use of parsing. Erbach's approach is promising because he extends Earley deduction for application of preference-based strategies. For that reason it seems interesting to combine his approach with that of $\mathcal{UTA}$. In [Johnson and Dörre, 1995] an Earley deduction mechanism is presented that uses a mechanism which is able to coroutine between goals that depend on each others' partial solutions. However, they only consider parsing. Den [1994] presents a chart-based algorithm based on Earley deduction that uses a similar agenda mechanism as $\mathcal{UTA}$, in particular he presents a cost-based abduction method used to

choose between alternative derivations. However, he only considers parsing, too.

Neither of the above mentioned approaches use shared items, basically because they do not consider interleaving of parsing and generation. Most approaches that consider an integrated approach can be found in the areas of artificial intelligence or cognitive science, e.g., [Jameson and Wahlster, 1982], [Vaughan and McDonald, 1986], [DeSmedt and Kempen, 1987], [Meteer and Shaked, 1988], [Levelt, 1989], [Wahlster *et al.*, 1991]. Neither of them however perform interleaving of parsing and generation with an comparable degree of granularity, nor do they consider uniform processing and item sharing.

7.2 Future extensions

Of course, the interleaved approach can be and should be extended and improved. Two of many possible ways which we started to investigate are briefly considered now.

Explanation-based learning An important line of research will be the application of explanation-based learning (EBL) to speed up processing. In [Neumann, 1994a] we have described the application of EBL to efficient parsing of constraint-based grammars. The idea is to generalize the derivations of training instances created by normal parsing automatically and to use these generalized derivations (also called templates) during the run-time mode of the system. In the case that a template can be instantiated for a new input, no further grammatical analysis is necessary. The approach is not restricted to the sentential level but can also be applied to arbitrary sub-sentential phrases, i.e., it is possible to handle substrings of an input by templates. Therefore, the EBL method can be interleaved straightforwardly with normal processing to get back flexibility that otherwise would be lost. In the work mentioned above we have shown how this interleaving is obtained by using an agenda-based Earley style parser.

With the existence of $\mathcal{UTA}$ we are now in a position to adapt the same method also to *generation*, and to interleave EBL and normal generation in the same way as we do it for parsing. Moreover, we are also able to extend the item sharing approach to yield a kind of *template sharing* approach—leading to *reversible EBL*.

Preferences Another very important line of future research will be the integration of *preference-based strategies*.

We have mentioned several times the importance of preferences for natural language processing and we have been careful to avoid obstacles to this important future direction. $\mathcal{UTA}$'s agenda mechanism, for example, is already an important pre-requisition for the incorporation of such strategies, since it allows processing of new items in any order. Also the architecture of the item sharing approach has been designed to support preference-based control.

The strategies described in [Uszkoreit, 1991] and [Barnett, 1994] seem to be suitable candidates for the new uniform environment. The work described in [Uszkoreit, 1991]

is of importance since the approach focusses on the integration of preferences with the feature system of a constraint-based grammar as an appropriate means for obtaining plausible performance models. In [Barnett, 1994] a model is described that is able to handle specific preferences for parsing and generation, as well as shared preferences.

It is reasonable to assume that both strategies (even together) can be integrated into the new uniform model. If so, it would also be possible to realize a sort of *preference-based monitoring* strategy. We assume that the NLS in which the uniform model is integrated maintains different preference spaces for parsing and generation. Preference-based monitoring would then mean that the derivation of a produced utterance is directed so that it is consistent with respect to the assumed preferences of the interlocutor which have been used to direct parsing (clearly, this presumes that the NLS has as its disposal a user and discourse model). For example, if both prefer minimal–attachment of pp-modifiers then an utterance like "Remove the folder with the system tools" (with meaning "Remove the folder by means of the system tools") would cause no revision.

8 Conclusion

We have developed a uniform computational model for natural language parsing and generation. It is based on a novel uniform tabular algorithm called $\mathcal{UTA}$ for parsing and generation from constraint-based grammars, and a new method of grammatical processing called item sharing. On the basis of these methods we have shown how an elegant but practical interleaving of parsing and generation is achieved by a novel incremental monitoring algorithm that is used during natural language production. These methods have been fully implemented in Common Lisp and CLOS.

Although uniformly defined $\mathcal{UTA}$ is fully driven by the structure of the actual input—a string for parsing and a semantic expression for generation. Since the only relevant parameter our algorithm has with respect to parsing and generation is the difference in input structures, the basic differences between parsing and generation are simply the different input structures. This seems to be trivial; however, our approach is the first uniform algorithm that is able to adapt its behaviour dynamically to the data, achieving *a maximal degree of uniformity of parsing and generation.* None of the current uniform approaches exhibit such a degree of uniformity.

There is evidence that comprehension and generation are not only inverse mappings, but that they are related to each other also at the processing level. For example, the human mechanism also involves some monitoring of the output and it is widely accepted that this is performed by making use of the comprehension mechanism. However, it has been an open question as to how such a behaviour can practically be realized in computer systems. We have paid serious attention to that problem, and we obtained as an answer that systematic pursuit of uniformity in natural language processing achieves the necessary preconditions for a practical interleaving of parsing and generation.

A A Complete Parsing and Generation Example Run with $\mathcal{UTA}$

We will use the following grammar fragment to illustrate the behaviour of $\mathcal{UTA}$:[15]

vp(Sem)←v(Sem) np pp
vp(Sem)←v(Sem) pp np
vp(Sem)←v(Sem) np
np(Sem)←n(Sem)
np(Sem)←np(Sem) pp
pp(Sem)←p(Sem) np

The phrasal backbone of this grammar is context-free. Thus we implicitly assume that strings are represented as difference lists which are simply concatenated. For parsing we can assume that the value of Ef is bound to ⟨PHON DL⟩ and for generation the value is bound to ⟨SEM⟩.

The figure 3 illustrates how $\mathcal{UTA}$ processes the string "sieht Peter mit Maria" ("sees Peter with Mary") during her parsing mode, and figure 4 shows the trace of the semantic expression "sehen(Peter,mit(Maria))" ("to_see(Peter,with(Mary))"). The simple grammar used has the nice property, that for the string "sieht Peter mit Maria" two readings "sehen(peter, mit(maria))" and "sehen(peter (mit(maria))" will be analysed and for the reading "sehen(peter, mit(maria))" the two strings "sieht Peter mit Maria" and "sieht mit Maria Peter" are generated. Thus the example illustrates very well how we can reuse completed structures in parsing as well as in generation.

[15] We do not claim that this fragment is linguistically adequate. Its sole function is to illustrate the behaviour of the uniform indexing mechanism.

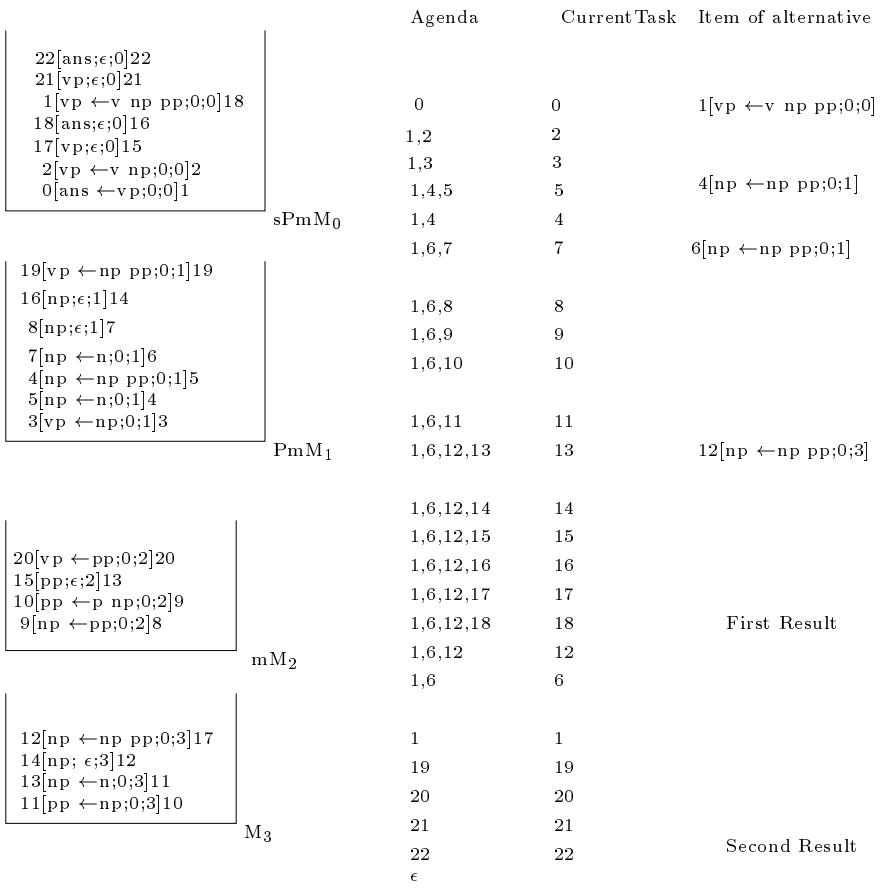

Figure 3: A trace through parsing of the string "sieht Peter mit Maria".

We assume that a lemma counter is used that enumerates the lemmas just created (starting from 0) and that the agenda mechanism selects tasks in a depth-first manner. We also count the items that have been placed in some item set starting by 1. The lemma counter will be attached to an item as a prefix, and the item counter as its suffix. To make things more readable, we are using only the initials of each word of the string. Thus "sPmM" abbreviates the string "sieht Peter mit Maria". The sequence in which item sets are created is indicated by using a counter starting from 0. Thus the index of the initial item set is "$sPmM_0$". The counter will then be used as an abbreviation for the item set indices in an item. We also show the status of the agenda and the current selected task. We also show those items which represent alternatives but are suspended in an extra row "Item of alternative", to make the depth-first strategy more readable.

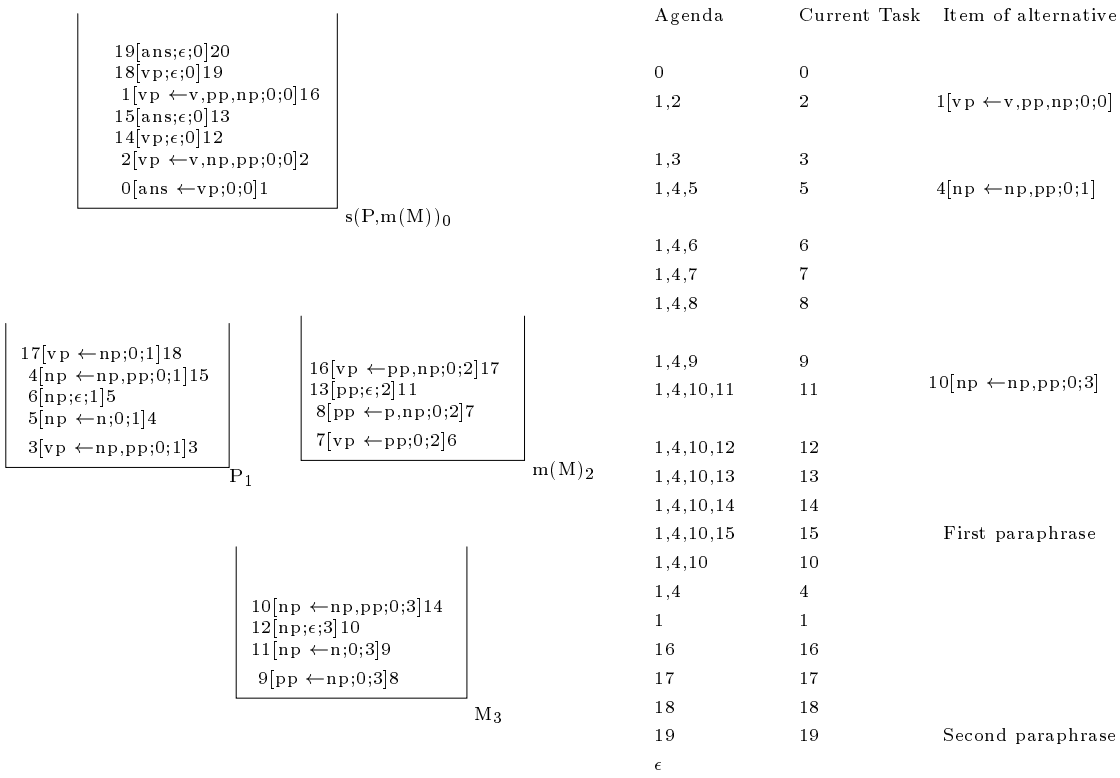

Figure 4: A trace through generation of "sehen(Peter,mit(Maria))".

We make use of the abbreviations introduced in the previous figure 3. Thus "s(P,m(M))" abbreviates the semantic expression "sehen(Peter, mit(Maria))". We also assume that the agenda control processes tasks in a depth-first manner. Note that we need to use the path ⟨SEM⟩ as essential feature. This is the only requirement to let $\mathcal{UTA}$ to run for generation in an efficient manner. The selection function "simulates" the semantic-head first selection function, although coincidentally in all cases the head element is located in leftmost position. The second paraphrase is generated by reusing the PP "mit Maria" (item 13) and the NP "Peter" (item 6) already computed during the generation of the first paraphrase. Since the item sets are indexed by means of semantic information, there is no problem in placing these strings at different string positions as for the first paraphrase. In this example, the item sets are created in sequentially because of the depth-first strategy. If we had used a breadth-first strategy, the item sets I_{P1} and $I_{m(M)_2}$ would have been created simultaneously.

References

[Alshawi and Crouch, 1992] H. Alshawi and R. Crouch. Monotonic semantic interpretation. In *30th Annual Meeting of the Association for Computational Linguistics*, Newark, Delaware, 1992.

[Appelt, 1987] D. E. Appelt. Bidirectional grammars and the design of natural language generation systems. In Y. Wilks, editor, *Theoretical Issues in Natural Language Processing-3*, pages 185–191. Hillsdale, N.J.: Erlbaum, 1987.

[Backofen and Weyers, 1993] R. Backofen and C. Weyers. $\mathcal{UDiNe}$—A Feature Constraint Solver with Distributed Disjunction and Classical Negation. Technical report, DFKI, Saarbrücken, Germany, 1993. Forthcoming.

[Barnett, 1994] J. Barnett. Bi-directional preferences. In Tomek Strzalkowski, editor, *Reversible Grammar in Natural Language Processing*, pages 201–234. Kluwer, 1994.

[Berg, 1986] T. Berg. The problems of language control: Editing, monitoring and feedback. *Psychological Research*, 48:133–144, 1986.

[Bresnan, 1982] J. Bresnan, editor. *The Mental Representation of Grammatical Relations.* MIT Press, 1982.

[Den, 1994] Y. Den. Generalized chart algorithm: An efficient procedure for cost-based abduction. In *32th Annual Meeting of the Association for Computational Linguistics*, New Mexico, 1994.

[DeSmedt and Kempen, 1987] K. DeSmedt and G. Kempen. Incremental sentence production, self–correction and coordination. In G. Kempen, editor, *Natural Language Generation*, pages 365–376. Martinus Nijhoff, Dordrecht, 1987.

[Earley, 1970] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

[Erbach, 1995] Gregor Erbach. *Bottom-Up Earley Deduction for Preference-Driven Natural Language Processing.* PhD thesis, Universität des Saarlandes, Germany, forthcoming 1995.

[Frazier, 1982] L. Frazier. Shared components of production and perception. In M. A. Arbib et al., editor, *Neural Models of Language Processes*, pages 225–236. Academic Press, New York, 1982.

[Gerdemann, 1991] D. D. Gerdemann. *Parsing and Generation of Unification Grammars.* PhD thesis, University of Illinois, Cognitive Science, Technical Report CS-91-06, 1991.

[Haas, 1989] A. Haas. A parsing algorithm for unification grammars. *Computational Linguistics*, 15(4):219–232, 1989.

[Höhfeld and Smolka, 1988] M. Höhfeld and G. Smolka. Definite relations over constraint languages. Technical Report Technical Report No. 53,LILOG IBM, Stuttgart, 1988.

[Jacobs, 1988] P. S. Jacobs. Achieving bidirectionality. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, pages 267–274, Budapest, 1988.

[Jaffar and Lassez, 1987] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages, Munich, Germany*, pages 111–119. ACM, January 1987.

[Jameson and Wahlster, 1982] A. Jameson and W. Wahlster. User modelling in anaphora generation: Ellipsis and definite description. In *Proceedings of teh 1982 European Conference on Artificial Intelligence*, pages 222–227, Orsay, France, 1982.

[Johnson and Dörre, 1995] M. Johnson and J. Dörre. Memoization of coroutined constraints. In *33th Annual Meeting of the Association for Computational Linguistics*, Cambridge, 1995.

[Kay, 1986] M. Kay. Algorithm schemata and data structures in syntactic processing. In B. J. Grosz, K. Sparck Jones, and B. L. Webber, editors, *Natural Language Processing*, pages 35–70. Kaufmann, Los Altos, CA, 1986.

[Keene, 1989] S. E. Keene. *Object-Oriented Programming in COMMON LISP: A Programmer's Guide to CLOS*. Addison-Wesley, Reading, MA, 1989.

[Kempen and Hoenkamp, 1987] G. Kempen and E. Hoenkamp. An incremental procedural grammar for sentence formulation. *Cognitive Science*, 11:201–258, 1987.

[Kempen, 1989] G. Kempen. Language generation systems. In I. S. Batori, W. Lenders, and W. Putschke, editors, *Computational Linguistics - Computerlinguistik*, pages 471–480. de Gruyter, Berlin, 1989.

[Levelt, 1989] W. J. M. Levelt. *Speaking: From Intention to Articulation*. MIT Press, Cambridge, Massachusetts, 1989.

[Levine, 1992] J. M. Levine. Pragma: A flexible bidirectional dialogue system. In *AAAI-90*, pages 964–969, Boston, 1992.

[Lloyd, 1987] J. .W. Lloyd. *Foundations of Logic Programming*. Symbol Computation, Springer, Berlin, New York, 1987.

[Meteer and Shaked, 1988] M. M. Meteer and V. Shaked. Strategies for effective paraphrasing. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, Budapest, 1988.

[Meteer, 1990] M. M. Meteer. *The Generation Gap – the problem of expressibility in text planning*. PhD thesis, University of Massachusetts, 1990.

[Neumann and Finkler, 1990] G. Neumann and W. Finkler. A head-driven approach to incremental and parallel generation of syntactic structures. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, pages 288–293, Helsinki, 1990.

[Neumann and van Noord, 1992] G. Neumann and G. van Noord. Self-monitoring with reversible grammars. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING)*, pages 700–706, Nantes, 1992.

[Neumann and van Noord, 1994] G. Neumann and G. van Noord. Reversibility and self-monitoring in natural language generation. In Tomek Strzalkowski, editor, *Reversible Grammar in Natural Language Processing*, pages 59–96. Kluwer, 1994.

[Neumann, 1994a] G. Neumann. Application of explanation-based learning for efficient processing of constraint-based grammars. In *Proceedings of the Tenth IEEE Conference on Artifical Intelligence for Applications*, pages 208–215, San Antonio, Texas, March 1994.

[Neumann, 1994b] Günter Neumann. *A Uniform Computational Model for Natural Language Parsing and Generation*. PhD thesis, Universität des Saarlandes, Germany, November 1994.

[Pereira and Shieber, 1987] F. C. N. Pereira and S. M. Shieber. *Prolog and Natural Language Analysis*. Center for the Study of Language and Information Stanford, 1987.

[Pereira and Warren, 1983] F. C. N. Pereira and D. Warren. Parsing as deduction. In *21st Annual Meeting of the Association for Computational Linguistics*, Cambridge Massachusetts, 1983.

[Pollard and Sag, 1994] C. Pollard and I. M. Sag. *Head-Driven Phrase Structure Grammar*. Center for the Study of Language and Information Stanford, 1994.

[Ristad, 1993] E. S. Ristad. *The Language Complexity Game*. MIT-Press, 1993.

[Robertson, 1994] S. P. Robertson. Tsunami: Simultaneous understanding, answering, and memory interactions for questions. *Cognitive Science*, 18:51–85, 1994.

[Samuelsson, 1994] C. Samuelsson. *Fast Natural-Language Parsing Using Explanation-Based Learning*. PhD thesis, Swedish Institute of Computer Science, Kista, Sweden, 1994.

[Shieber *et al.*, 1983] S. M. Shieber, H. Uszkoreit, F. C. N. Pereira, J. Robinson, and M. Tyson. The formalism and implementation of PATR-II. In B. J. Grosz and M. E. Stickel, editors, *Research on Interactive Acquisition and Use of Knowledge*. SRI report, 1983.

[Shieber *et al.*, 1990] S. M. Shieber, F. C. N. Pereira, G. van Noord, and R. C. Moore. Semantic-head-driven generation. *Computational Linguistics*, 16(1), 1990.

[Shieber, 1985] S. M. Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *23th Annual Meeting of the Association for Computational Linguistics*, Chicago, 1985.

[Shieber, 1988] S. M. Shieber. A uniform architecture for parsing and generation. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, Budapest, 1988.

[Shieber, 1989] S. M. Shieber. *Parsing and Type Inference for Natural and Computer Languages*. PhD thesis, Stanford University, SRI International Technical note 460, 1989.

[Shieber, 1993] S. M. Shieber. The problem of logical-form equivalence. *Computational Linguistics*, 19:179–190, 1993.

[Smolka, 1988] G. Smolka. A feature logic with subsorts. Technical report, IBM Deutschland GmbH, Germany, 1988. Lilog-Report 33.

[Smolka, 1992] G. Smolka. Feature constraint logics for unification grammars. *The Journal of Logic Programming*, 12:51–87, 1992.

[Somers *et al.*, 1990] H. Somers, J. Tsujii, and D. Jones. Machine translation without a source text. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, volume 3, pages 271–276, Helsinki, 1990.

[Steele, 1990] G. L. Steele. *Common LISP: The Language (Second Edition)*. Digital Press, Burlington, MA, 1990.

[Strzalkowski, 1994] T. Strzalkowski. A general computational method for grammar inversion. In Tomek Strzalkowski, editor, *Reversible Grammar in Natural Language Processing*, pages 175–199. Kluwer, 1994.

[Uszkoreit, 1991] H. Uszkoreit. Strategies for adding control information to declarative grammars. In *29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, 1991.

[VanNoord, 1993] G. J. M. VanNoord. *Reversibility in Natural Language Processing.* PhD thesis, University of Utrecht, The Netherlands, 1993.

[Vaughan and McDonald, 1986] M. M. Vaughan and D. D. McDonald. A model of revision in natural language generation. In *24th Annual Meeting of the Association for Computational Linguistics*, pages 90–96, 1986.

[Wahlster *et al.*, 1991] W. Wahlster, E. André, W. Graf, and T. Rist. Designing illustrated texts: How language production is influenced by graphics generation. In *Fifth Conference of the European Chapter of the Association for Computational Linguistics*, pages 8–14, Berlin, 1991.

[Wahlster, 1991] W. Wahlster. User and discourse models for multimodal communication. In *Intelligent user interfaces*, chapter 3, pages 45–67. ACM Press, 1991.

[Winston and Horn, 1989] P. H. Winston and B. K. P. Horn. *LISP: Third Edition.* Addison-Wesley, Reading, MA, 1989.

[Wirén and Rönnquist, 1993] Mats Wirén and Ralph Rönnquist. Fully Incremental Parsing. In *Proc. Third International Workshop on Parsing Technologies*, Tilburg, the Netherlands and Durbuy, Belgium, 1993.